Logical Methods in Computer Science
Vol. 7 (2:6) 2011, pp. 1–32
www.lmcs-online.org

Submitted May 29, 2010
Published May 5, 2011

AUTOMATED SYNTHESIS OF TABLEAU CALCULI [*]

RENATE A. SCHMIDT AND DMITRY TISHKOVSKY

School of Computer Science, The University of Manchester, UK
e-mail address: {renate.schmidt,dmitry.tishkovsky}@manchester.ac.uk

Abstract. This paper presents a method for synthesising sound and complete tableau calculi. Given a specification of the formal semantics of a logic, the method generates a set of tableau inference rules that can then be used to reason within the logic. The method guarantees that the generated rules form a calculus which is sound and constructively complete. If the logic can be shown to admit finite filtration with respect to a well-defined first-order semantics then adding a general blocking mechanism provides a terminating tableau calculus. The process of generating tableau rules can be completely automated and produces, together with the blocking mechanism, an automated procedure for generating tableau decision procedures. For illustration we show the workability of the approach for a description logic with transitive roles and propositional intuitionistic logic.

1. Introduction

Tableau-based reasoning is popular in many areas of computer science and various branches of logic. For description logics and ontology reasoning they provide the main method for doing reasoning (see, for example, [5, 24]; some recent work is [32, 30]). For modal logics and applications such as multi-agent systems tableau approaches are frequently used (see, for example, [18, 13, 21, 29, 14, 23]; some recent work is [20, 6]). Tableau calculi have been developed and are being used for non-classical logics such as intuitionistic logic [18, 3], conditional logic [2], logics of metric and topology [27] and hybrid logics [37, 10, 15]. Rather than developing tableau calculi one by one for individual logics, it is possible to develop tableau calculi in a systematic way for large classes of logics. This is evident from the literature in all these areas and studies such as [22, 17, 3].

In this paper we want to go further and investigate the possibility to generate tableau calculi automatically from the specification of a logic. We assume that the logic of interest is defined by a high-level specification of its formal semantics. Our aim is to turn this into a set of inference rules that provides a sound and complete deduction calculus for the logic. Ideally we also want to be able to guarantee termination if the logic is decidable. Automated synthesis of calculi is a challenging problem and in general it is of course not possible to turn every specification of a logic into a sound, complete and terminating deduction calculus.

1998 ACM Subject Classification: F.4.1, I.2.3, I.2.4.

Key words and phrases: calculus synthesis, tableau calculi, soundness, completeness, decidability, description logic, modal logic, first-order logic, automated reasoning.

[*] The paper is an extended and improved version of [34].

It is however possible to describe classes of logical specifications for which the problem is solvable uniformly.

In previous work we have shown that it is possible to synthesise tableau calculi for modal logics by translation to first-order logic combined with first-order resolution [31]. In this approach the semantic specification of a logic is transformed into clausal form and then a set of inference rules. Soundness and completeness of the generated calculus follows from the soundness and completeness of the simulating resolution refinement used. In the present paper we introduce another approach for generating tableau calculi. Rather than proceeding via simulation by resolution, our approach generates a tableau calculus directly from the specification of a logic. For traditional modal logics essentially the same tableau calculi can be obtained, but for more expressive dynamic modal logics and description logics the method in [31] produces calculi with introduction rules, whereas the method in this paper can be used to produce calculi with only elimination rules.

In other previous work we have described a framework for turning sound and complete tableau calculi into decision procedures [33]. The key for this framework is the unrestricted blocking mechanism from [32] which is added to the given calculus in order to turn it into a terminating calculus. Enhancing a tableau calculus with the unrestricted blocking mechanism produces a terminating tableau calculus, whenever the logic can be shown to admit finite filtration with respect to its semantics [33]. More specifically, the prerequisites are that the following conditions all hold.

(1) The logic admits the effective finite model property shown by a filtration argument.
(2) The tableau calculus is sound and constructively complete.
(3) A weak form of subexpression property holds for tableau derivations.

Constructive completeness is a slightly stronger notion than completeness and means that for every open branch in a tableau there is a model which reflects all the expressions (formulae) occurring on the branch. The subexpression property says that every expression in a derivation is a subexpression of the input expression with respect to a finite subexpression closure operator.

In order to be able to exploit this 'termination through blocking' result from [33], in this paper, our goal is to synthesise tableau calculi that satisfy the prerequisites (2) and (3). It turns out that, provided the specification of the semantics of the logic is well-defined in a certain sense, the subexpression property can be imposed on the generated calculus. Crucial is the separation of the syntax of the logic from the 'extras' in the meta-language needed for the semantic specification of the logic. The process of generating tableau calculi can be completely automated and gives, together with the unrestricted blocking mechanism and the results in [32, 33], an automated procedure for generating tableau decision procedures for logics, whenever they have the effective finite model property with respect to a well-defined first-order semantics, that is, condition (1) holds.

The tableau synthesis method introduced in this paper works as follows. The user defines the formal semantics of the given logic in a many-sorted first-order language so that certain well-definedness conditions hold. The semantic specification of the logic is then automatically reduced to Skolemised implicational forms which are further transformed into tableau inference rules. Combined with a set of default closure and equality rules, the generated rules provide a sound and constructively complete calculus for the logic. Under certain conditions the set of rules can be further refined. If the logic can be shown to admit finite filtration, then the generated calculus can be automatically turned into a terminating calculus by adding the unrestricted blocking mechanism from [32].

The method is intended to be as general as possible, and cover as many logics as possible. Our main focus is non-classical logics and description logics. As case studies we consider the application of the method to propositional intuitionistic logic IPC [28] and the description logic $\mathcal{SO}$. Propositional intuitionistic logic provides a nearly perfect example because the semantics of the logical connectives is not Boolean and the semantics is restricted by a background theory. In addition, the logic is simple. $\mathcal{SO}$ is the extension of the description logic $\mathcal{ALC}$ with singleton concepts (or nominals) and transitive roles. $\mathcal{SO}$ is a fragment of many expressive description logics considered in the literature [4] and is the analogue of the hybrid [9] version of the standard modal logic K4 [8].

The paper is structured as follows. Section 2 defines the apparatus for specifying the logic of interest. It consists of two languages, a language for specifying the syntax of the logic and a language for specifying its semantics. How to specify the semantics of a logic is described in Section 3. Because there are many ways of writing semantic specifications, in this paper, we focus on what we call well-defined semantic specifications for which sound and complete tableau calculi can be generated. The tableau generation process is presented in Section 4, and Section 5 proves soundness and constructive completeness of the generated calculus. Sections 6 and 7 discuss two techniques for refining a calculus. The first refinement aims at reducing branching in derivations. The second refinement aims at reducing the use of extraneous constructs in the language of the tableau calculus. In Section 8 we show how the unrestricted blocking mechanism of [32] can be used to obtain terminating tableau calculi for logics with the effective finite model property. To illustrate the approach we use the description logic $\mathcal{SO}$ as a running example throughout the paper. In Section 9 the approach is applied to propositional intuitionistic logic. The paper concludes with a discussion of the approach.

The paper is written using terminology of description logics, but all the results apply equally to modal logics and other non-classical logics. In most cases where we use the word 'expression' we could have equally used the words 'formula' or 'logical term'.

2. The Specification Languages

In order for the user to specify the semantics of the given logic for which they want to develop a tableau calculus there are two specification languages:

(1) an object language for defining the syntax of the logic, and
(2) a meta-language for specifying the semantics of the logic.

For the sake of generality the *object language*, denoted by $\mathcal{L}$, is a many-sorted propositional language, thus allowing for the specification of many-sorted propositional logics including modal logics, description logics and other non-classical logics.

Throughout the paper the standard notation ω is used for the smallest infinite countable ordinal, that is, $\omega = \{0, 1, 2, \ldots\}$.

Let $\mathit{Sorts} \stackrel{\text{def}}{=} \{0, 1, \ldots, N\}$ be the index set of the sorts of the object language. The idea is that, for $n = 1, \ldots, N$, symbols of sort n are interpreted as n-ary relations and symbols of sort 0 are interpreted as domain elements. Of the sorts, the sort 1 is regarded as the *primary sort.*

Let Conn be a countable set of the logical connectives of the logic to be specified. Every connective σ in Conn is associated with a tuple $(i_1, i_2, \ldots, i_{m+1}) \in \mathit{Sorts}^{(m+1)}$, where $m \geq 0$.

The last argument i_{m+1} is the sort of the expression obtained by applying σ to expressions of sorts $i_1, i_2, \ldots, i_m$, respectively. We say that σ is an m-ary connective of sort $(i_1, i_2, \ldots, i_{m+1})$.

The object language $\mathcal{L}$ is defined over an alphabet given by a set of sorts Sorts, a set of connectives Conn, a countable set of variable symbols $\{p^i_j \mid i \in \mathsf{Sorts}, j \in \omega\}$, and a countable set of constant symbols $\{q^i_j \mid i \in \mathsf{Sorts}, j \in \omega\}$. $\mathcal{L}$ is defined as the set of *expressions* over the alphabet closed under the connectives in Conn. More formally, let $\mathcal{L} \overset{\mathsf{def}}{=} \bigcup_{i \in \mathsf{Sorts}} \mathcal{L}^i$, where each $\mathcal{L}^i$ denotes the *set of expressions of sort* i defined as the smallest set of expressions satisfying the following conditions:

- All variables p^i_j and all constants q^i_j in the alphabet are expressions belonging to $\mathcal{L}^i$.
- For every connective $\sigma \in \mathsf{Conn}$ of sort $(i_1, i_2 \ldots, i_{m+1})$, $\sigma(E_1, \ldots, E_m)$ is an expression belonging to $\mathcal{L}^{i_{m+1}}$, whenever $E_1, \ldots, E_m$ belong to $\mathcal{L}^{i_1}, \ldots, \mathcal{L}^{i_m}$, respectively.

Symbols, expressions and connectives in the language $\mathcal{L}$ are also referred to as $\mathcal{L}$-symbols, $\mathcal{L}$-expressions and $\mathcal{L}$-connectives. Variables and constants in $\mathcal{L}$ are called *atomic* $\mathcal{L}$-expressions. We refer to expressions in $\mathcal{L}^0$ as *individuals*, expressions in $\mathcal{L}^1$ as *concepts*, and expressions in $\mathcal{L}^2$ as *roles*. That is, individuals are expressions of sort 0, concepts are expressions (or formulae) of the primary sort and roles are expressions (or formulae) of sort 2.

For an $\mathcal{L}$-expression E, the notation $E(p_1, \ldots, p_m)$ indicates that $p_1, \ldots, p_m$ are (distinct) variables occurring in the expression E. To avoid ambiguity in this notation we standardly assume that all the variables of the language $\mathcal{L}$ are linearly ordered by an ordering $<_v$ and $p_1 <_v \cdots <_v p_m$. $E(E_1, \ldots, E_m)$ denotes the expression obtained by uniformly substituting E_i into p_i, for all $i = 1, \ldots, m$. Similarly, if X is a set of $\mathcal{L}$-expressions depending on variables $p_1, \ldots, p_m$, we indicate this as $X(p_1, \ldots, p_m)$ and denote by $X(E_1, \ldots, E_m)$ the set of expressions which are instances of expressions from X under uniform substitution of the expressions $E_1, \ldots, E_m$ into $p_1, \ldots, p_m$, respectively.

Throughout the paper we use the logic $\mathcal{SO}$ as a running example. Recall that $\mathcal{SO}$ is the description logic $\mathcal{ALC}$ extended with nominals, or singleton concepts, and transitive roles.

The object language $\mathcal{L}_{\mathcal{SO}}$ for specifying the syntax of $\mathcal{SO}$ consists of three sorts, namely 0, 1 and 2 for individuals, concepts, and roles, respectively. Atomic expressions of sort 0 are individual variables from a countable set $\{p^0_j \mid j \in \omega\}$. We denote individual variables also by $\ell_0, \ell_1, \ldots$. The variables p^1_j are of sort 1 and are the concept symbols. In this paper concept symbols are denoted by $p_0, p_1, \ldots$. The variables p^2_j of sort 2 are the atomic roles and are denoted by $r_0, r_1, \ldots$.

The connectives in $\mathcal{L}_{\mathcal{SO}}$ are the following:

- The 'singleton concept' connective $\{\cdot\}$ of the sort $(0, 1)$. That is, $\{\ell\}$ is a concept for every individual ℓ.
- The Boolean connectives $\sqcup$ and $\neg$ of sorts $(1, 1, 1)$ and $(1, 1)$, respectively. As usual, we use infix notation for $\sqcup$ and prefix notation for $\neg$. Thus, $C \sqcup D$ and $\neg C$ are concepts for any concept expressions C and D.
- The existential restriction connective $\exists \cdot . \cdot$ of sort $(2, 1, 1)$. That is, $\exists r.C$ is a concept for any role expression r and concept expression C.

Thus, expressions of $\mathcal{L}_{\mathcal{SO}}$ are built from individuals, concept symbols and role symbols using the given connectives, and there are no other expressions in $\mathcal{L}_{\mathcal{SO}}$. In this language, individual and role expressions are allowed to be atomic only.

The *meta-language* in which the semantics of the given logic is specified is a many-sorted first-order language with equality and is denoted by $\mathsf{FO}(\mathcal{L})$. $\mathsf{FO}(\mathcal{L})$ extends the object language $\mathcal{L}$, the idea being that $\mathcal{L}$-expressions are represented as terms in $\mathsf{FO}(\mathcal{L})$ and $\mathcal{L}$-connectives as functions.

Formally, $\mathsf{FO}(\mathcal{L})$ is defined as an extension of $\mathcal{L}$ with one additional sort, namely $N+1$, additional symbols, the standard first-order connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, the equivalence connective $\equiv$, first-order quantifiers $\exists$ and $\forall$, and the equality predicate $\approx$. Thus the sorts of $\mathsf{FO}(\mathcal{L})$ are $\mathsf{Sorts} \cup \{N+1\} = \{0, \ldots, N, N+1\}$. We call the additional sort $N+1$ the *domain sort*, and symbols over this sort are called the *domain symbols*. The additional symbols comprise of a countable set of variable symbols $\{x, y, z, x_0, y_0, z_0, \ldots\}$ of the domain sort, a countable set of constants $\{a, b, c, a_0, b_0, c_0, \ldots\}$ of the domain sort, function symbols $\{f, g, h, f_0, g_0, h_0, \ldots\}$ mapping argument terms to terms of sort $N+1$, and a countable set of constant predicate symbols $\{P, Q, R, P_0, Q_0, R_0, \ldots\}$ of the domain sort (that is, argument terms are required to be terms of sort $N+1$). Intuitively, the domain sort contains symbols necessary for formalising semantic properties of the domain elements of interpretations of the target logic.

In addition, $\mathsf{FO}(\mathcal{L})$ contains the symbols $\nu_0, \ldots, \nu_N$, one for each sort in Sorts of the object language. In particular, ν_0 is a unary function symbol of sort $(0, N+1)$ (that is, a function from sort 0 to sort $N+1$). Each of the remaining ν_i is a predicate symbol of sort $(i, N+1, \ldots, N+1)$ with arity $i+1$.

The purpose of these symbols is to define the semantics of the connectives of the logic by using conditions similar to satisfaction conditions in standard definitions. ν_0 can be viewed as the interpretation mapping for individuals (represented as terms) in the object language. All other ν_n can be viewed as interpretation mappings for expressions in the object language; they can be viewed as the 'holds' or 'satisfaction' predicates.

Finally, for every sort we assume the presence in $\mathsf{FO}(\mathcal{L})$ of a binary predicate symbol representing the equality predicate for that sort. For reasons of simplicity, we use one symbol, namely $\approx$, for each of the equality predicates.

Formulae in $\mathsf{FO}(\mathcal{L})$ are just first-order formulae defined over the symbols of $\mathsf{FO}(\mathcal{L})$, where each expression in $\mathcal{L}$ is represented by a term in $\mathsf{FO}(\mathcal{L})$. In particular, each variable symbol p^i_j in $\mathcal{L}^i$ is represented by a variable of sort i in $\mathsf{FO}(\mathcal{L})$, each constant symbol q^i_j in $\mathcal{L}^i$ is represented by a constant of sort i in $\mathsf{FO}(\mathcal{L})$, and every connective σ is represented by a function of the same sort as σ.

To illustrate how expressions of a logic are represented in a meta-language we continue our running example. According to our definitions the meta-language $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$ for $\mathcal{SO}$ is a first-order language with sorts 0, 1, 2 and 3. The interpretation symbols are ν_0 (which is a function symbol) and the holds predicate symbols ν_1 and ν_2. Also included in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$ is the equality predicate symbol $\approx$.

Every variable of $\mathcal{L}_{\mathcal{SO}}$ is represented by a variable of the corresponding sort in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$. Thus, every individual variable ℓ in $\mathcal{L}_{\mathcal{SO}}$ is represented by a variable of sort 0 in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$. Every concept symbol p in $\mathcal{L}_{\mathcal{SO}}$ is represented by a variable of sort 1, and every role symbol r in $\mathcal{L}_{\mathcal{SO}}$ by a variable of sort 2 in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$. Connectives of the object language become function symbols of an appropriate sort in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$. Thus, every expression in $\mathcal{L}_{\mathcal{SO}}$ becomes a first-order term of the corresponding sort. For instance, the concept expression $\exists r.p$ is represented as a term of sort 1.

Whereas the sorts 0, 1, and 2 are the sorts in the object language $\mathcal{L}_{\mathcal{SO}}$, the sort 3 is a separate sort in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$ with its own sets of variables, individual constants, function symbols, and symbols of predicate constants. Sort 3 is the domain sort for $\mathcal{SO}$.

Finally, for every individual ℓ, $\nu_0(\ell)$ is a term of sort 3 in $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$, and $\nu_1(C,t)$ and $\nu_2(r,t,t')$ are atomic formulae of $\mathsf{FO}(\mathcal{L}_{\mathcal{SO}})$, for any concept expression C, any role expression r, and any terms t and t' of sort 3.

Before we describe how a logic can be defined in the meta-language $\mathsf{FO}(\mathcal{L})$ in the next section, we fix some more notation and terminology. Let $\overline{w}$ denote a sequence of first-order variables, that is $\overline{w}\overset{\mathsf{def}}{=}w_1,\ldots,w_n$. Similarly, let $\forall\overline{w}$ denote the universal quantifier prefix on all variables $w_1,\ldots,w_n$, that is, $\forall\overline{w}\overset{\mathsf{def}}{=}\forall w_1\cdots\forall w_n$. For any set S of formulae, $\forall S$ denotes the universal closure of S, that is, the set

$$\forall S \overset{\mathsf{def}}{=} \{\forall\overline{w}\,\phi(\overline{w}) \mid \phi(\overline{w}) \in S\}.$$

For every first-order formula ψ we let

$$\sim\psi \overset{\mathsf{def}}{=} \begin{cases}\psi', & \text{provided } \psi = \neg\psi',\\ \neg\psi, & \text{otherwise.}\end{cases}$$

Formulae of $\mathsf{FO}(\mathcal{L})$ in which all occurrences of the $\mathcal{L}$-variables p^i_j (of sorts $i = 0,\ldots,N$) are free are called *$\mathcal{L}$-open* formulae. An *$\mathcal{L}$-open sentence* is an $\mathcal{L}$-open formula that does *not* have free occurrences of variables of the domain sort $N+1$.

For example, the formula

$$\forall y\,(\nu_1(\exists r.p, y) \wedge \nu_2(r,x,y))$$

is an $\mathcal{L}_{\mathcal{SO}}$-open formula because the variables p and r occur only freely. Because the variable x of domain sort 3 also occurs freely, it is *not* an $\mathcal{L}_{\mathcal{SO}}$-open sentence. In contrast, the formula

$$\forall y\,(\nu_1(\exists r.p, y) \wedge \forall x\,\nu_2(r,x,y))$$

is an $\mathcal{L}_{\mathcal{SO}}$-open sentence, because all the occurrences of the domain variables x and y are bound by quantifiers and all the occurrences of p and r are unbound. The formulae

$$\forall p\forall y\,(\nu_1(\exists r.p, y) \wedge \nu_2(r,x,y)) \quad \text{and} \quad \forall r\,(\nu_1(\exists r.p, y) \wedge \nu_2(r,x,y))$$

are not $\mathcal{L}_{\mathcal{SO}}$-open because of the presence of quantified variables of sorts other than the domain sort (p and r). (The symbol $\exists$ in $\exists r.p$ should not be confused with the existential quantifier of first-order logic.)

For any set S of $\mathcal{L}$-open formulae in $\mathsf{FO}(\mathcal{L})$ and a set X of $\mathcal{L}$-expressions, let

$$S{\restriction}X \overset{\mathsf{def}}{=} \{\phi(E_1,\ldots,E_m) \mid \phi(p_1,\ldots,p_m) \in S \text{ and } \\ \text{all } \mathcal{L}\text{-expressions occurring in } \phi(E_1,\ldots,E_m) \text{ belong to } X\}.$$

$S{\restriction}X$ is the set of instances of formulae in S under substitutions into the variables of $\mathcal{L}$ that do not contain expressions outside X.

Suppose, for example,

$$S \overset{\mathsf{def}}{=} \{\nu_1(\exists r.p, y), \nu_1(\neg p, x)\} \quad \text{and} \quad X \overset{\mathsf{def}}{=} \{r_0, p_0, p, p \sqcap p_0, \exists r_0.p_0\}.$$

Then the instantiations of formulae in S relative to X are

$$\nu_1(\exists r_0.p_0, x),\ \nu_1(\exists r_0.p, x),\ \nu_1(\exists r_0.p \sqcap p_0, x),\ \nu_1(\exists r_0.\exists r_0.p_0, x),$$
$$\nu_1(\neg p_0, x),\ \nu_1(\neg p, x),\ \nu_1(\neg(p \sqcap p_0), x),\ \nu_1(\neg\exists r_0.p_0, x).$$

The only formula in this list where all $\mathcal{L}$-subexpressions belong to X is $\nu_1(\exists r_0.p_0, y)$. Thus

$$S{\restriction}X = \{\nu_1(\exists r_0.p_0, y)\}.$$

The formula $\nu_1(\exists r_0.p, y)$ does not belong to $S{\restriction}X$ because $\exists r_0.p$ does not belong to X. Other instances do not belong to $S{\restriction}X$ for similar reasons.

3. Specifying the Semantics of an Object Language

First, we define the model structures in terms of which the semantics of the object language is then defined.

An *$\mathcal{L}$-structure* is a tuple $\mathcal{I} \overset{\mathsf{def}}{=} (\mathcal{L}^0, \ldots, \mathcal{L}^N, \Delta^\mathcal{I}, \nu_0^\mathcal{I}, \ldots, \nu_N^\mathcal{I}, a^\mathcal{I}, \ldots, P^\mathcal{I}, \ldots)$ where $\Delta^\mathcal{I}$ is a non-empty set, $\nu_0(\ell)^\mathcal{I} \in \Delta^\mathcal{I}$ for every individual $\ell \in \mathcal{L}^0$, $\nu_n^\mathcal{I} \subseteq \mathcal{L}^n \times (\Delta^\mathcal{I})^n$, for $0 < n \leq N$. $a^\mathcal{I} \in \Delta^\mathcal{I}$ and $P^\mathcal{I} \subseteq (\Delta^\mathcal{I})^m$, where m is the arity of P. For simplicity we omit the sets $\mathcal{L}^0, \ldots, \mathcal{L}^N$ and simply write

$$\mathcal{I} = (\Delta^\mathcal{I}, \nu_0^\mathcal{I}, \ldots, \nu_N^\mathcal{I}, a^\mathcal{I}, \ldots, P^\mathcal{I}, \ldots).$$

Observe that an $\mathcal{L}$-structure $\mathcal{I}$ is a first-order interpretation of the language $\textsf{\textit{FO}}(\mathcal{L})$.

For our sample logic $\mathcal{SO}$ an $\mathcal{L}_{\mathcal{SO}}$-structure is given by a tuple $\mathcal{I} = (\Delta^\mathcal{I}, \nu_0^\mathcal{I}, \nu_1^\mathcal{I}, \nu_2^\mathcal{I})$. This means, the $\nu_i^\mathcal{I}$ are arbitrary interpretation functions for $\mathcal{SO}$-expressions. As yet no additional conditions are assumed. In the description logic literature instead of a family of holds relations ν_i just one holds relation ν is used, resulting in the simpler and more familiar notation for an interpretation, namely $\mathcal{I} = (\Delta^\mathcal{I}, \nu^\mathcal{I})$.

A *valuation* in $\mathcal{I}$ is a mapping ι from the set of variables and constants of $\textsf{\textit{FO}}(\mathcal{L})$ to $\mathcal{L} \cup \Delta^\mathcal{I}$ such that $\iota(p_j^i), \iota(q_j^i) \in \mathcal{L}^i$, and $\iota(x_j), \iota(a_j) \in \Delta^\mathcal{I}$. We use the standard notation $\mathcal{I}, \iota \models \phi$ to indicate a (first-order) formula ϕ is *true* in the (first-order) interpretation $\mathcal{I}$ under valuation ι. Given a set of formulae S, we write $\mathcal{I}, \iota \models S$ if $\mathcal{I}, \iota \models \phi$ for every formula ϕ in S.

We say that a valuation ι in an $\mathcal{L}$-structure is *canonical* if every variable and constant of any sort $i = 0, \ldots, N$ is mapped to itself, that is, $\iota(p_j^i) = p_j^i$ and $\iota(q_j^i) = q_j^i$ for every variable p_j^i and constant q_j^i in the language $\mathcal{L}$. This means that the canonical valuation of any term of sort $i = 0, \ldots, N$ is the term itself.

It is not difficult to see that any $\mathcal{L}$-open formula ϕ is satisfiable in an $\mathcal{L}$-structure iff it is satisfiable in an $\mathcal{L}$-structure under a canonical valuation.

We write $S \models_c S'$ for sets of formulae S and S', if, for every $\mathcal{L}$-structure $\mathcal{I}$ and a canonical valuation ι in $\mathcal{I}$, $\mathcal{I}, \iota \models S$ implies $\mathcal{I}, \iota \models S'$. Similarly, we write $\mathcal{I} \models_c S$ iff there is a canonical valuation ι such that $\mathcal{I}, \iota \models S$.

Satisfiability for expressions of the given logic is defined only for expressions of the primary sort, that is, concept expressions. We say a concept expression C is *satisfiable* in $\mathcal{I}$ if there is an element a in $\Delta^\mathcal{I}$ such that $(C, a) \in \nu_1^\mathcal{I}$, or equivalently, $\mathcal{I} \models_c \exists x\ \nu_1(C, x)$. A concept expression C is *valid* in $\mathcal{I}$ if $\mathcal{I} \models_c \forall x\ \nu_1(C, x)$.

Next we describe how the semantics of a given logic can be specified in $\textsf{\textit{FO}}(\mathcal{L})$, where $\mathcal{L}$ is the object language of the logic.

$$\forall x\,(x \approx x) \qquad \forall x \forall y\,(x \approx y \to y \approx x) \qquad \forall x \forall y \forall z\,(x \approx y \wedge y \approx z \to x \approx z)$$
$$\forall x_1 \cdots \forall x_n \forall y_i\ (P(x_1, \dots, x_n) \wedge x_i \approx y_i \to P(x_1, \dots x_{i-1}, y_i, x_{i+1}, x_n))$$
$$\forall p\, \forall x_1 \cdots \forall x_n \forall y_i\ (\nu_n(p, x_1, \dots, x_n) \wedge x_i \approx y_i \to \nu_n(p, x_1, \dots x_{i-1}, y_i, x_{i+1}, x_n))$$
$$\forall p_1 \cdots \forall p_m \forall x_1 \cdots \forall x_n \forall y_i\ (x_i \approx y_i \to$$
$$f(p_1, \dots, p_m, x_1, \dots, x_n) \approx f(p_1, \dots, p_m, x_1, \dots x_{i-1}, y_i, x_{i+1}, \dots, x_n))$$

Figure 1: Default equality axioms in $\mathit{FO}(\mathcal{L})$.

Let S be any set of $\mathcal{L}$-open sentences in $\mathit{FO}(\mathcal{L})$ and σ be a connective of a sort $(i_1, \dots, i_m, n)$. A formula ϕ^σ in the language of S *defines the connective* σ with respect to S if it does not contain σ and the following holds:

$$\forall S \models \forall p_1 \dots \forall p_m\, \forall \overline{x}\ (\nu_n(\sigma(p_1, \dots, p_m), \overline{x}) \equiv \phi^\sigma(p_1, \dots, p_m, \overline{x})). \tag{3.1}$$

Here $p_1, \dots, p_m$ are variables of sorts $i_1, \dots, i_m$ respectively. If there is a formula ϕ^σ which defines σ with respect to S, we also say S *defines* σ and

$$\forall \overline{x}\ (\nu_n(\sigma(p_1, \dots, p_m), \overline{x}) \equiv \phi^\sigma(p_1, \dots, p_m, \overline{x})),$$

which is an $\mathcal{L}$-open sentence, is a *σ-definition with respect to S*. Connective definitions are always $\mathcal{L}$-open sentences, that is, they do not contain any quantifiers over variables of sorts $0, \dots, N$ (these are implicitly regarded as being universally quantified).

By definition, a *(first-order) semantic specification* of the object language $\mathcal{L}$ is a set S of $\mathcal{L}$-open $\mathit{FO}(\mathcal{L})$-sentences defining the connectives of $\mathcal{L}$. For the sake of generality we always include the standard equality axioms listed in Figure 1 in a semantic specification S. This ensures that $\approx$ is a congruence on every sort in any first-order interpretation of $\mathit{FO}(\mathcal{L})$. We assume the set of σ-definitions with respect to S of all the connectives σ of $\mathcal{L}$ is fixed and explicitly given as the set S^0.

Intuitively, a specification S of a semantics of the given logic is an axiomatisation in the language $\mathit{FO}(\mathcal{L})$ of a class of $\mathcal{L}$-structures where each $\mathcal{L}$-connective σ has an unambiguous representation. Because the Beth definability property holds for first-order logic we can assume that all such representations are explicit, that is, every connective σ is defined by an explicit formula ϕ^σ. The collection of explicit definitions of all the connectives constitutes the set S^0. Since there are many ways of axiomatising the same (axiomatisable) class of first-order structures and choosing explicit representations for connectives, there are many ways of specifying a semantics and choosing a set of semantic definitions for a semantic specification. Axiomatisations of the empty class of $\mathcal{L}$-structures are all inconsistent and, hence, semantic specifications can be inconsistent.

As an example we give a semantic specification for the logic $\mathcal{SO}$. Suppose $S_{\mathcal{SO}}$ consists of the following $\mathcal{L}_{\mathcal{SO}}$-open sentences together with the default equality axioms.

Connective definitions:

$$\forall x\ \big(\nu_1(\{\ell\}, x) \equiv \nu_0(\ell) \approx x\big)$$
$$\forall x\ \big(\nu_1(\neg p, x) \equiv \neg \nu_1(p, x)\big)$$
$$\forall x\ \big(\nu_1(p \sqcup q, x) \equiv \nu_1(p, x) \vee \nu_1(q, x)\big)$$
$$\forall x\ \big(\nu_1(\exists r.p, x) \equiv \exists y\ \big(\nu_2(r, x, y) \wedge \nu_1(p, y)\big)\big)$$

Transitivity axiom:

$$\forall x \forall y \forall z\ \big((\nu_2(r,x,y) \land \nu_2(r,y,z)) \rightarrow \nu_2(r,x,z)\big)$$

The first four sentences are the connective definitions of $\mathcal{L}_{\mathcal{SO}}$ and constitute the set $S^0_{\mathcal{SO}}$. The fifth sentence does not belong to $S^0_{\mathcal{SO}}$. It is the transitivity axiom specifying that all role symbols r are transitive. If we wanted to specify that only a subset of the role symbols are transitive, this can be done by including one transitivity axiom for each role (constant) symbol that is meant to be transitive.

Because, in general, there are many possibilities of axiomatising the same class of $\mathcal{L}$-structures, there are many possibilities for specifying the semantics of a logic. In this paper we restrict our attention to semantic specifications in forms that are standard in the literature for non-classical logics.

We say a semantic specification S is *normalised*, if it consists of three disjoint parts, that is, $S = S^+ \cup S^- \cup S^b$, where S^+, S^- and S^b are disjoint sets of sentences satisfying the following:

(n1) S^+ is a set of $\mathcal{L}$-open sentences of the form:

$$\xi^E_+ \overset{\mathsf{def}}{=} \forall \overline{x}\ (\nu_n(E(p_1,\ldots,p_m),\overline{x}) \rightarrow \phi^E_+(p_1,\ldots,p_m,\overline{x})).$$

(n2) S^- is a set of $\mathcal{L}$-open sentences of the form:

$$\xi^E_- \overset{\mathsf{def}}{=} \forall \overline{x}\ (\phi^E_-(p_1,\ldots,p_m,\overline{x}) \rightarrow \nu_n(E(p_1,\ldots,p_m),\overline{x})).$$

(n3) All $\mathcal{L}$-expressions occurring in S^b are atomic.

Here, E denotes any $\mathcal{L}$-expression.

In this definition we assume that multiple sentences of the form (n1) for the same expression E in S^+ and S^- are all equivalently reduced to a single sentence ξ^E_+. Similarly for (n2) and ξ^E_-. The intuition is that S^+ and S^- define the semantics of the connectives. S^+ defines it for positive occurrences of expressions E (with free variables $p_1,\ldots,p_m$), while S^- defines it for negative occurrences of expressions E. We refer to S^b as the *background theory* of the semantics S. In particular, S^b includes the equality axioms from Figure 1.

A semantic specification in the form $S^0 \cup S^b$ can be turned into normalised form by decomposing each connective definition in S^0 into two implications. In fact, S^0 and $S^+ \cup S^-$ play the same role in axiomatising $\mathcal{L}$-connectives in $\mathsf{FO}(\mathcal{L})$ modulo the background theory S^b.

The sample semantic specification $S_{\mathcal{SO}}$ can be normalised by decomposing the connective definitions in $S^0_{\mathcal{SO}}$ into $S^+_{\mathcal{SO}}$-sentences and $S^-_{\mathcal{SO}}$-sentences as follows.

$S^+_{\mathcal{SO}}$-sentences:

$$\begin{gathered}
\forall x\ \big(\nu_1(\{\ell\},x) \rightarrow \nu_0(\ell) \approx x\big) \\
\forall x\ \big(\nu_1(\neg p,x) \rightarrow \neg\nu_1(p,x)\big) \\
\forall x\ \big(\nu_1(p \sqcup q,x) \rightarrow \nu_1(p,x) \lor \nu_1(q,x)\big) \\
\forall x\ \big(\nu_1(\exists r.p,x) \rightarrow \exists y\ \big(\nu_2(r,x,y) \land \nu_1(p,y)\big)\big)
\end{gathered}$$

$S^-_{\mathcal{SO}}$-sentences:

$$\begin{aligned}
\forall x\ \big(\nu_0(\ell) \approx x \rightarrow \nu_1(\{\ell\}, x)\big)\\
\forall x\ \big(\neg\nu_1(p, x) \rightarrow \nu_1(\neg p, x)\big)\\
\forall x\ \big(\nu_1(p, x) \vee \nu_1(q, x) \rightarrow \nu_1(p \sqcup q, x)\big)\\
\forall x\ \big(\exists y\ \big(\nu_2(r, x, y) \wedge \nu_1(p, y)\big) \rightarrow \nu_1(\exists r.p, x)\big)
\end{aligned}$$

The background theory $S^b_{\mathcal{SO}}$ of $\mathcal{SO}$ consists of this sentence,

$$\forall x \forall y \forall z\ \big((\nu_2(r, x, y) \wedge \nu_2(r, y, z)) \rightarrow \nu_2(r, x, z)\big),$$

specifying transitivity of roles plus the default equality axioms.

It is worth noting that the symbol E in definitions (n1) and (n2) denotes an arbitrary expression in $\mathcal{L}$. This means that E does not necessarily have the form $\sigma(p_1, \ldots, p_n)$ where σ is a connective. For example, a specification might be:

$$\xi^E_+ \overset{\mathsf{def}}{=} \forall x\ (\nu_1(\exists r.\exists r.p, x) \rightarrow \nu_1(\exists r.p, x))$$

In this case $E \overset{\mathsf{def}}{=} \exists r.\exists r.p$ and $\phi^E_+ \overset{\mathsf{def}}{=} \nu_1(\exists r.p, x)$.

It is convenient to introduce notation for the set of instantiations of the right hand sides and left hand sides of the ξ^E_+ and ξ^E_-, respectively. For every $\mathcal{L}$-expression E, let

$$\begin{aligned}
\Phi^E_+ &\overset{\mathsf{def}}{=} \{\phi^F_+(E_1, \ldots, E_m, \overline{x}) \mid E = F(E_1, \ldots, E_m) \text{ for some } \xi^{F(p_1,\ldots,p_m)}_+ \text{ from } S\} \text{ and}\\
\Phi^E_- &\overset{\mathsf{def}}{=} \{\phi^F_-(E_1, \ldots, E_m, \overline{x}) \mid E = F(E_1, \ldots, E_m) \text{ for some } \xi^{F(p_1,\ldots,p_m)}_- \text{ from } S\}.
\end{aligned}$$

Thus, Φ^E_+ (respectively Φ^E_-) is the set of instantiations of succedents (respectively antecedents) of positive (respectively negative) specifications in S, where the antecedents (respectively succedents) match the given expression E.

For example, in the case of our specification for $\mathcal{SO}$ and $E = \exists r.(p \sqcup q)$, we have

$$\Phi^{\exists r.(p\sqcup q)}_+ = \Phi^{\exists r.(p\sqcup q)}_- = \{\exists y\ (\nu_2(r, x, y) \wedge \nu_1(p \sqcup q, y))\}.$$

Let $\prec$ be any ordering on $\mathcal{L}$-expressions. For any $\mathcal{L}$-expression E and any set X of $\mathcal{L}$-expressions we define

$$\mathsf{sub}_\prec(E) \overset{\mathsf{def}}{=} \{E' \mid E' \prec E\} \quad \text{and} \quad \mathsf{sub}_\prec(X) \overset{\mathsf{def}}{=} \bigcup_{E \in X} \mathsf{sub}_\prec(E).$$

That is, $\mathsf{sub}_\prec(X)$ is the set of all expressions $\prec$-smaller than some expression in X. We often write $\mathsf{sub}_\prec(E_1, \ldots, E_m)$ rather than $\mathsf{sub}_\prec(\{E_1, \ldots, E_m\})$.

Any normalised specification S of a semantics *induces* a relation $\prec$ on expressions as follows. Let $\prec$ be the smallest transitive relation satisfying: $E' \prec E$ whenever $E = F(E_1, \ldots, E_m)$, for some $\mathcal{L}$-expressions $E_1, \ldots, E_m$, and E' occurs in $\phi^F_+(E_1, \ldots, E_m, \overline{x})$ or $\phi^F_-(E_1, \ldots, E_m, \overline{x})$, respectively, for some sentence $\xi^{F(p_1,\ldots,p_m)}_+$ or $\xi^{F(p_1,\ldots,p_m)}_-$ in S. The reflexive closure of $\prec$ is denoted by $\preceq$.

Recall that S^0 denotes the set of $\mathcal{L}$-open sentences that define the $\mathcal{L}$-connectives. A semantic specification S is *well-defined* iff S is normalised and the following conditions are all true.

(wd1) $\forall S^0, \forall S^b \models \forall S$,

(wd2) the relation $\prec$ induced by S is a well-founded ordering on $\mathcal{L}$-expressions, and (wd3) for every expression $E = \sigma(E_1, \ldots, E_m)$,

$$\forall S^0, S^b{\restriction}\mathsf{sub}_{\prec}(E) \models_c \forall\overline{x}\Big(\Big(\bigwedge \Phi^E_+ \to \phi^\sigma(E_1, \ldots, E_m, \overline{x})\Big) \wedge \Big(\phi^\sigma(E_1, \ldots, E_m, \overline{x}) \to \bigvee \Phi^E_-\Big)\Big).$$

Condition (wd3) follows from the following first-order condition:

(wd3′) for every connective σ,

$$\forall S^0, S^b{\restriction}\mathsf{sub}_{\prec}(\sigma(\overline{p})) \models_c \forall\overline{x}\Big(\Big(\bigwedge \Phi^{\sigma(\overline{p})}_+ \to \phi^\sigma(\overline{p}, \overline{x})\Big) \wedge \Big(\phi^\sigma(\overline{p}, \overline{x}) \to \bigvee \Phi^{\sigma(\overline{p})}_-\Big)\Big).$$

Because we can assume that S^0 is also a normalised semantic specification, it similarly induces a relation $\prec_0$ that can be assumed to be a well-founded ordering. Standardly, the semantics of a logic is defined by induction over the interpretation of the connectives and primitives (that is, constants, and variables) which is homomorphically lifted to arbitrary $\mathcal{L}$-expressions. This is equivalent to assuming a well-founded ordering on expressions of $\mathcal{L}$. For any reasonable definition of a semantics such a well-founded ordering exists. Thus, although it is not difficult to imagine formulae ϕ^σ such that $\prec_0$ is not well-founded, we assume that the ϕ^σ are chosen in such a way that it is possible to lift the semantics of $\mathcal{L}$-primitives to arbitrary $\mathcal{L}$-expressions, that is, $\prec_0$ is well-founded.

In the case of $S_{\mathcal{SO}}$, because $S^+_{\mathcal{SO}}$ and $S^-_{\mathcal{SO}}$ are obtained by decomposing the set $S^0_{\mathcal{SO}}$, the two orderings $\prec$ and $\prec_0$ coincide. Similar to many cases of description and modal logics, $\prec$ and $\prec_0$ are both just the direct subexpression ordering on $\mathcal{L}_{\mathcal{SO}}$.

There are different semantic specifications which describe the same class of $\mathcal{L}$-structures. As we have just noted, some semantic specifications already allow the lifting of the semantics from atomic expressions to arbitrary $\mathcal{L}$-expressions. We assume that $S^0 \cup S^b$ is such a specification and implicitly accommodates $\mathcal{L}$-connectives. According to this definition, a well-defined semantic specification S is equivalent to $S^0 \cup S^b$ modulo the background theory S^b. This is ensured by condition (wd1) and the assumption that S defines all $\mathcal{L}$-connectives in S^0. Through condition (wd2), S imposes its own inductive structure on $\mathcal{L}$-expressions. Condition (wd3) specifies a correlation between S and S^0 on instances of $\mathcal{L}$-expressions. It can be seen that $S^0 \cup S^b$ is a well-defined semantic specification itself.

Let us consider if the semantic specification of $\mathcal{SO}$ above is well-defined. The first condition is satisfied because $S_{\mathcal{SO}} = S^0_{\mathcal{SO}} \cup S^b_{\mathcal{SO}}$. The second condition is satisfied because $\prec$ is the direct subexpression ordering. Condition (wd3′) is true for all $\mathcal{SO}$ connectives. For instance, consider the case of $\sigma = \exists \cdot .\cdot$. Since $\Phi^{\exists r.p}_+ = \Phi^{\exists r.p}_- = \{\exists y\,(\nu_2(r, x, y) \wedge \nu_1(p, y))\}$, the formula

$$\forall x\,\Big(\big(\exists y\,(\nu_2(r, x, y) \wedge \nu_1(p, y)) \to \phi^\sigma(r, p, x)\big) \wedge \big(\phi^\sigma(r, p, x) \to \exists y\,(\nu_2(r, x, y) \wedge \nu_1(p, y))\big)\Big),$$

on the right hand side of condition (wd3′), is a tautology. In a similar way, the condition (wd3′) can be checked for the other connectives.

A *(propositional) logic* L over the language $\mathcal{L}$ is a subset of concepts in $\mathcal{L}$ which is closed under arbitrary substitutions of variables with expressions of the same sorts. A logic L is *first-order definable* iff there is a semantic specification S_L such that L coincides with the set of all concepts that are valid in all $\mathcal{L}$-structures satisfying $\forall S_L$, that is,

$$L = \{C \in \mathcal{L}^1 \mid \forall S_L \models_c \forall x\, \nu_1(C, x)\}.$$

For a fixed semantic specification S_L of a logic L, if $\mathcal{I}$ is an $\mathcal{L}$-structure satisfying S_L then by definition $\mathcal{I}$ is a *model of L* or simply an *L-model* (with respect to S_L).

4. Synthesising a Tableau Calculus

First, we give the needed basic definitions for the kind of tableau calculi our method generates.

Let T denote a tableau calculus comprising of a set of inference rules. A *tableau derivation* or *tableau* for T is a finitely branching, ordered tree whose nodes are sets of formulae in $\mathsf{FO}(\mathcal{L})$. Assuming that $\mathcal{S}$ is the input set of concept expressions in $\mathcal{L}$ to be tested for satisfiability the root node of the tableau is the set $\{\nu_1(C, a) \mid C \in \mathcal{S}\}$, where a denotes a fresh constant of the domain sort. For a finite set $\mathcal{S}$, a can be viewed as the Skolem constant introduced by Skolemising the $\mathsf{FO}(\mathcal{L})$-formula $\exists x \ \bigwedge_{C\in\mathcal{S}} \nu_1(C, x)$. (This can be naturally expanded to infinite sets of concepts but this is not essential for the paper.)

Successor nodes are constructed in accordance with a set of inference rules in the calculus. The inference rules have the general form

$$\frac{X_0}{X_1|\dots|X_n},$$

where both the numerator X_0 and all denominators X_i are finite sets of negated or unnegated atomic formulae in the language $\mathsf{FO}(\mathcal{L})$. The formulae in the numerator are called *premises*, while the formulae in the denominators are called *conclusions*. n is called the *branching factor* of the rule. The numerator and all the denominators are non-empty, but n may be zero, in which case the denominators are not present and the rule is a *closure rule*. Closure rules are also written $X_0/\bot$. If the branching factor n is greater than one, the rule is a *branching rule*. An inference rule is applicable to a selected formula ϕ in a node of the tableau, if ϕ together with other formulae in the node, are simultaneous instantiations of all the premises of the rule. Then n successor nodes are created which contain the formulae of the current node and the appropriate instances of X_i. We assume that *any rule is applied at most once to the same set of premises*, which is a standard assumption for tableau derivations.

We use the notation $T(\mathcal{S})$ for a finished (in the limit) tableau built by applying the rules of the calculus T starting with the set $\mathcal{S}$ (of $\mathcal{L}$-concepts) as input. That is, we assume that all branches in the tableau are fully expanded and all applicable rules have been applied in $T(\mathcal{S})$. We assume that all the rules of the calculus are applied *non-deterministically to a tableau.* This means that we do not assume any order of rule application and, at any given node, an arbitrary rule is chosen for the node expansion from all the rules which are applicable to formulae of the node.

In a tableau, a maximal path from the root node is called a *branch*. For a branch $\mathcal{B}$ of a tableau we write $\phi \in \mathcal{B}$ to indicate that the formula ϕ has been derived in $\mathcal{B}$, that is, ϕ belongs to a node of the branch $\mathcal{B}$. Our notion of a tableau branch can be viewed in two ways. On the one hand, one can view it as having procedural flavour as a path of nodes in the tableau derivation. On the other hand, a branch can be identified with the set-theoretical union of the nodes in it.

A branch of a tableau is *closed* if a closure rule has been applied in this branch, otherwise the branch is called *open*. The tableau $T(\mathcal{S})$ is *closed* if all its branches are closed and $T(\mathcal{S})$ is *open* otherwise. The calculus T is *sound* iff for any (possibly infinite) set of concepts $\mathcal{S}$, each $T(\mathcal{S})$ is open whenever $\mathcal{S}$ is satisfiable. T is *complete* iff for any (possibly infinite) unsatisfiable set of concepts $\mathcal{S}$ there is a $T(\mathcal{S})$ which is closed.

Now, let L be a first-order definable propositional logic over $\mathcal{L}$ and S_L a well-defined semantic specification of L, that is, conditions (wd1)–(wd3) hold for S_L. We now describe how tableau rules can be synthesised from S_L. If S_L is not already normalised we first normalise it. Thus assume $S_L = S_L^+ \cup S_L^- \cup S_L^b$.

Next we take a positive specification ξ_+^E in S_L^+. Eliminate existential quantifiers using Skolemisation and equivalently rewrite ξ_+^E into the following implicational form

$$\forall x_1 \cdots \forall x_n \ \left(\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n) \rightarrow \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk} \right),$$

where each ψ_{jk} denotes a literal. This is always possible. The implication is now turned into the rule:

$$\rho_+(\xi_+^E) \ \overset{\mathsf{def}}{=} \ \frac{\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n), \quad y_1 \approx y_1, \quad \ldots, \quad y_s \approx y_s}{\psi_{11}, \quad \ldots, \quad \psi_{1K_1} \ \mid \cdots \mid \ \psi_{J1}, \quad \ldots, \quad \psi_{JK_J}},$$

where $y_1, \ldots, y_s$ denote the free variables occurring in ψ_{jk} which do not occur among the variables $x_1, \ldots, x_n$. Essentially, the antecedent of the implication has become the main premise in the numerator and the succedent has been turned into the denominators of the rule. We say the rule *corresponds* to ξ_+^E. This is repeated for each positive specification in S_L^+.

Analogously, we generate a tableau rule for each negative specification ξ_-^E in S_L^-. The corresponding rules have the form

$$\rho_-(\xi_-^E) \ \overset{\mathsf{def}}{=} \ \frac{\neg\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n), \quad y_1 \approx y_1, \quad \ldots, \quad y_s \approx y_s}{\psi_{11}, \quad \ldots, \quad \psi_{1K_1} \ \mid \cdots \mid \ \psi_{J1}, \quad \ldots, \quad \psi_{JK_J}}.$$

This is obtained by Skolemising the contrapositive of ξ_-^E and then equivalently rewriting it to an implication of the form

$$\forall x_1 \cdots \forall x_n \ \left(\neg\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n) \rightarrow \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk} \right),$$

where each ψ_{jk} denotes a literal.

We refer to the rules $\rho_+(\xi_+^E)$ and $\rho_-(\xi_-^E)$ generated in this way, as the *decomposition rules.*

If the right hand sides of the implicational forms contain free variables y_i then these are assumed to be universally quantified and the generated rules are γ-rules in the Smullyan classification. Our use of the equalities $y_i \approx y_i$ in the premises of the generated rules is a bit non-standard, and can be omitted if this is preferred. We use the equalities to achieve domain predication, which makes explicit that applying γ-rules only instantiates with terms (domain elements) that occur on the current branch.

The sentences in the background theory of S_L are turned into rules by first equivalently transforming them into Skolemised disjunctive normal form. More specifically, let ξ be an

$$\frac{P(x_1,\dots,x_n)}{x_1\approx x_1,\quad \dots,\quad x_n\approx x_n} \qquad \frac{\neg P(x_1,\dots,x_n)}{x_1\approx x_1,\quad \dots,\quad x_n\approx x_n}$$

$$\frac{\nu_n(p,x_1,\dots,x_n)}{p\approx p,\quad x_1\approx x_1,\quad \dots,\quad x_n\approx x_n} \qquad \frac{\neg\nu_n(p,x_1,\dots,x_n)}{p\approx p,\quad x_1\approx x_1,\quad \dots,\quad x_n\approx x_n}$$

$$\frac{x\approx y}{y\approx x} \qquad \frac{x\approx y,\quad y\approx z}{x\approx z}$$

$$\frac{P(x_1,\dots,x_n),\quad x_i\approx y_i}{P(x_1,\dots,x_{i-1},y_i,x_{i+1},\dots,x_n)} \qquad \frac{\nu_n(p,x_1,\dots,x_n),\quad x_i\approx y_i}{\nu_n(p,x_1,\dots,x_{i-1},y_i,x_{i+1},\dots,x_n)}$$

$$\frac{f(p_1,\dots,p_m,x_1,\dots,x_n)\approx f(p_1,\dots,p_m,x_1,\dots,x_n),\quad x_i\approx y_i}{f(p_1,\dots,p_m,x_1,\dots,x_n)\approx f(p_1,\dots,p_m,x_1,\dots x_{i-1},y_i,x_{i+1},\dots,x_n)}$$

Figure 2: Default equality rules for predicates and functions occurring in S_L.

arbitrary sentence in S^b_L. It is first equivalently rewritten to

$$\forall x_1\cdots\forall x_n \bigvee_{j=1}^{J}\bigwedge_{k=1}^{K_j}\psi_{jk}(p_1,\dots,p_m,x_1,\dots,x_n), \tag{4.1}$$

where each ψ_{jk} denotes a literal, and is then turned into the corresponding rule, namely

$$\rho(\xi)\ \stackrel{\mathsf{def}}{=}\ \frac{p_1\approx p_1,\quad \dots,\quad p_m\approx p_m,\quad x_1\approx x_1,\quad \dots,\quad x_n\approx x_n}{\psi_{11},\quad \dots,\quad \psi_{1K_1}\ \mid\ \cdots\ \mid\ \psi_{J1},\quad \dots,\quad \psi_{JK_J}}.$$

The $p_1,\dots,p_m,x_1,\dots,x_n$ are the variables appearing in (4.1). The purpose of the equalities in the premises is domain predication and can optionally be omitted. Rules corresponding to sentences in S^b_L are called *theory rules*.

For example, the generated decomposition rules for the existential restriction operator in the description logic $\mathcal{SO}$ are

$$\frac{\nu_1(\exists r.p,x)}{\nu_2(r,x,f(r,p,x)),\quad \nu_1(p,f(r,p,x))} \quad\text{and}\quad \frac{\neg\nu_1(\exists r.p,x),\quad y\approx y}{\neg\nu_2(r,x,y)\ \mid\ \neg\nu_1(p,y)}.$$

$f(r,p,x)$ in the left rule is the Skolem term introduced for the quantifier $\exists y$ in the connective definition of $\exists\cdot.\cdot$. The intuition is that for each r, each p and each x matching the premise of the rule there is an element $f(r,p,x)$ so that the conclusions of the rule are both true. The transitivity property for roles in the background theory of the semantic specification of $\mathcal{SO}$ is transformed to the rule

$$\frac{r\approx r,\quad x\approx x,\quad y\approx y,\quad z\approx z}{\neg\nu_2(r,x,y)\ \mid\ \neg\nu_2(r,y,z)\ \mid\ \nu_2(r,x,z)}.$$

These rules are not the familiar rules used in standard description logic tableau systems, but in Section 6 we see how to get those by rule refinement.

The *equality rules* are generated in essentially the same way from the equality axioms in the background theory and are refined in accordance with the method described in Section 6. Figure 2 lists the full set of the *refined* equality rules included by default in the generated tableau calculus.

Since in our formalisation the equality predicate(s) are also used as domain predicate(s) in order to keep track of the ground terms that occur in the tableau branches, we include rules which ensure that expressions of the form $t\approx t$ are treated as domain predicates and appear in every branch of a tableau for every term t in the branch. These are the first four rules in Figure 2. In particular, these rules ensure that for any term occurring in a literal $(\neg)P(t_1,\ldots,t_n)$ or $(\neg)\nu_n(q,t_1,\ldots,t_n)$ on any branch, the equalities $t_1 \approx t_1, \ \ldots, \ t_n \approx t_n$ and $q \approx q$ are added to the branch. The rules also state reflexivity of the equality predicate(s). The remaining rules are variations of standard rules for equality. The rules in row three and four ensure that $\approx$ is a congruence relation for predicates on terms occurring in a branch. The rule in the last row is a congruence rule for function symbols f occurring in a branch including Skolem function symbols.

We note that the equality predicate $\approx$ is treated as an ordinary constant predicate symbol of the meta-language $\mathit{FO}(\mathcal{L})$ and, hence, can occur in any place where an ordinary predicate symbol P can occur.

Finally the generated tableau calculus also includes the following *closure rules.*

$$\frac{\nu_n(p,\overline{x}), \quad \neg\nu_n(p,\overline{x})}{\bot} \qquad\qquad \frac{P(\overline{x}), \quad \neg P(\overline{x})}{\bot} \tag{4.2}$$

for each sort n and every constant predicate symbol P occurring in the semantic specification S_L of the logic.

We use T_L to denote the generated tableau calculus. In summary, it consists of these rules.

(t1) The decomposition rules $\rho^\sigma_+(\xi)$ and $\rho^\sigma_-(\xi')$ corresponding to all positive specifications ξ in S^+_L and all negative specifications ξ' in S^-_L.
(t2) The theory rules $\rho(\zeta)$ corresponding to all sentences ζ in the background theory S^b_L.
(t3) The equality rules of Figure 2.
(t4) The closure rules (4.2).

Note for each connective there are exactly two decomposition rules in the calculus T_L, one for unnegated occurrences and one for negated occurrences of the connective.

For $\mathcal{SO}$ the described approach generates the tableau rules listed in Figure 3.

5. Ensuring Soundness and Constructive Completeness

We first prove soundness of the calculus T_L synthesised from a normalised semantic specification S_L. It is possible to prove that every rule of the generated calculus T_L preserves satisfiability of $\mathit{FO}(\mathcal{L})$-formulae. That is, if all premises of a rule are true in an L-model $\mathcal{I}$ (under a canonical valuation) then the conclusions of some branch are also true. This is the case because the transformation of the semantic specification ensures that the definitions of the rules basically mimic the semantic definitions. Hence, soundness is ensured.

Theorem 5.1 (Soundness). *Let T_L be a tableau calculus generated from a normalised semantic specification S_L of a logic L. Then T_L is sound for L, that is, for every possibly infinite set of concepts $\mathcal{S}$ satisfiable in an L-model, any finished tableau derivation $T_L(\mathcal{S})$ is open.*

Now, we prove constructive completeness of T_L. Let $\mathcal{B}$ denote an arbitrary branch in a T_L-tableau derivation. We define the following relation $\sim_\mathcal{B}$ with respect to $\mathcal{B}$:

$$t \sim_\mathcal{B} t' \overset{\mathsf{def}}{\iff} t \approx t' \in \mathcal{B},$$

Decomposition rules:

$$\frac{\nu_1(\{\ell\},x)}{\nu_0(\ell)\approx x}\qquad\frac{\neg\nu_1(\{\ell\},x)}{\nu_0(\ell)\not\approx x}\qquad\frac{\nu_1(\neg p,x)}{\neg\nu_1(p,x)}\qquad\frac{\neg\nu_1(\neg p,x)}{\nu_1(p,x)}$$

$$\frac{\nu_1(p_1\vee p_2,x)}{\nu_1(p_1,x)\mid\nu_1(p_2,x)}\qquad\frac{\neg\nu_1(p_1\vee p_2,x)}{\neg\nu_1(p_1,x),\quad\neg\nu_1(p_2,x)}$$

$$\frac{\nu_1(\exists r.p,x)}{\nu_2(r,x,f(r,p,x)),\quad\nu_1(p,f(r,p,x))}\qquad\frac{\neg\nu_1(\exists r.p,x),\quad y\approx y}{\neg\nu_2(r,x,y)\mid\neg\nu_1(p,y)}$$

Transitivity rule:

$$\frac{r\approx r,\quad x\approx x,\quad y\approx y,\quad z\approx z}{\neg\nu_2(r,x,y)\mid\neg\nu_2(r,y,z)\mid\nu_2(r,x,z)}$$

Equality congruence rules:

$$\frac{x\approx y}{x\approx x,\quad y\approx y}\qquad\frac{x\not\approx y}{x\approx x,\quad y\approx y}\qquad\frac{x\approx y}{y\approx x}\qquad\frac{x\approx y,\quad y\approx z}{x\approx z}$$

$$\frac{\nu_1(p,x)}{p\approx p,\quad x\approx x}\qquad\frac{\neg\nu_1(p,x)}{p\approx p,\quad x\approx x}\qquad\frac{\nu_2(r,x,y)}{r\approx r,\quad x\approx x,\quad y\approx y}\qquad\frac{\neg\nu_2(r,x,y)}{r\approx r,\quad x\approx x,\quad y\approx y}$$

$$\frac{\nu_1(p,x),\quad x\approx y}{\nu_1(p,y)}\qquad\frac{\nu_2(r,x,y),\quad x\approx z}{\nu_2(r,z,y)}\qquad\frac{\nu_2(r,x,y),\quad y\approx z}{\nu_2(r,x,z)}$$

$$\frac{f(r,p,x)\approx f(r,p,x),\quad x\approx y}{f(r,p,x)\approx f(r,p,y)}$$

Closure rules:

$$\frac{\nu_1(p,x),\quad\neg\nu_1(p,x)}{\bot}\qquad\frac{\nu_2(r,x,y),\quad\neg\nu_2(r,x,y)}{\bot}\qquad\frac{x\approx y,\quad x\not\approx y}{\bot}$$

Figure 3: Generated tableau rules for $\mathcal{SO}$.

for any ground terms t and t' of the domain sort $N+1$ in $\mathcal{B}$. Let $\|t\|\overset{\mathsf{def}}{=}\{t'\mid t\sim_{\mathcal{B}}t'\}$ be the equivalence class of an element t. The presence of the rules of Figure 2 ensures that $\sim_{\mathcal{B}}$ is a congruence relation on all domain ground terms in $\mathcal{B}$.

We say a model $\mathcal{I}$, under a (canonical) valuation ι, *reflects* an expression E of the sort n occurring in a branch $\mathcal{B}$ iff for all ground terms $t_1,\ldots,t_n$ we have that

- $(E,\iota(t_1),\ldots,\iota(t_n))\in\nu_n^{\mathcal{I}}$ whenever $\nu_n(E,t_1,\ldots,t_n)\in\mathcal{B}$, and
- $(E,\iota(t_1),\ldots,\iota(t_n))\notin\nu_n^{\mathcal{I}}$ whenever $\neg\nu_n(E,t_1,\ldots,t_n)\in\mathcal{B}$.

Similarly, $\mathcal{I}$ *reflects* predicate constant P from $\mathcal{B}$ under a (canonical) valuation ι in $\mathcal{I}$ iff for all ground terms $t_1,\ldots,t_n$ we have that

- $(\iota(t_1),\ldots,\iota(t_n))\in P^{\mathcal{I}}$ whenever $P(t_1,\ldots,t_n)\in\mathcal{B}$, and
- $(\iota(t_1),\ldots,\iota(t_n))\notin P^{\mathcal{I}}$ whenever $\neg P(t_1,\ldots,t_n)\in\mathcal{B}$.

A model $\mathcal{I}$ *reflects* branch $\mathcal{B}$ under a valuation ι, if $\mathcal{I}$ reflects all predicate constants and expressions occurring in $\mathcal{B}$ under ι.

A tableau calculus T_L is said to be *constructively complete* (for L) iff for any given set of concept $\mathcal{S}$, if $\mathcal{B}$ is an open branch in a tableau derivation $T_L(\mathcal{S})$ then there is an L-model $\mathcal{I}$ such that:

(m1) The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is the set of the equivalence classes $\|t\|$ for each ground term t occurring in $\mathcal{B}$.
(m2) $\mathcal{I}$ reflects $\mathcal{B}$ under the *canonical projection valuation* π defined by $\pi(t) \overset{\mathsf{def}}{=} \|t\|$, for every ground term t occurring in $\mathcal{B}$.

It is clear that if T_L is constructively complete then T_L is complete for L.

Suppose now that S_L is a well-defined semantic specification and $\prec_0$ is a well-founded ordering on $\mathcal{L}$-expressions induced by the set S_L^0 of the definitions of the connectives of the form (3.1) with respect to S_L.

Let $\mathcal{B}$ be an open branch in a finished tableau derivation in T_L. We define interpretations of predicate symbols in $\mathcal{I}(\mathcal{B})$ by induction on $\prec_0$ as follows:

- For every n-ary constant predicate symbol P in S_L,
$$P^{\mathcal{I}(\mathcal{B})} \overset{\mathsf{def}}{=} \{(\|t_1\|, \dots, \|t_n\|) \mid P(t_1, \dots, t_n) \in \mathcal{B}\}.$$
- For every $n = 1, \dots, N$ the interpretation $\nu_n^{\mathcal{I}(\mathcal{B})}$ of the ν_n symbols is defined as the smallest subset of $\mathcal{L}^n \times (\Delta^{\mathcal{I}(\mathcal{B})})^n$ satisfying both the following, for every variable or constant p of the sort n, every connective σ, and any expressions $E_1, \dots, E_m$:
$$\begin{aligned}&(p, \|t_1\|, \dots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})} \iff \nu_n(p, t_1, \dots, t_n) \in \mathcal{B},\\ &(\sigma(E_1, \dots, E_m), \|t_1\|, \dots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})}\\ &\qquad\qquad \iff \mathcal{I}(\mathcal{B}) \models_c \phi^\sigma(E_1, \dots, E_m, \|t_1\|, \dots, \|t_n\|).\end{aligned}$$

In what follows, we say that $\mathcal{I}(\mathcal{B})$ reflects an expression E (a predicate P, or a branch $\mathcal{B}$) if $\mathcal{I}(\mathcal{B})$ reflects E (P, or $\mathcal{B}$, respectively) under the canonical projection valuation π, and omit any explicit reference to π.

A consequence of the definition of $\mathcal{I}(\mathcal{B})$ is that the definitions of the connectives are valid in $\mathcal{I}(\mathcal{B})$:

Lemma 5.2. $\mathcal{I}(\mathcal{B}) \models \forall S_L^0$.

Lemma 5.3. *Let X be any set of expressions occurring in $\mathcal{B}$. Suppose $\mathcal{I}(\mathcal{B})$ reflects all the expressions from X. Then $\mathcal{I}(\mathcal{B}) \models_c S_L^b \restriction X$.*

Proof. Consider any $\xi \in S_L^b$ and suppose the Skolemised form of ξ is as in (4.1), that is:
$$\xi(p_1, \dots, p_m) \equiv \forall x_1 \cdots \forall x_n \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk}(p_1, \dots, p_m, x_1, \dots, x_n).$$
Let $E_1, \dots, E_m$ be any expressions from X and $t_1, \dots, t_n$ any ground terms of sort $N+1$ occurring in $\mathcal{B}$. By rule $\rho(\xi)$, there is a $j = 1, \dots, J$ such that, for all $k = 1, \dots, K_j$, the literals $\psi_{jk}(E_1, \dots, E_m, t_1, \dots, t_n)$ are in the branch $\mathcal{B}$. Since S_L^b does not contain non-atomic expressions of the language $\mathcal{L}$ we have that $\mathcal{I}(\mathcal{B}) \models_c \psi_{jk}(E_1, \dots, E_m, \|t_1\|, \dots, \|t_n\|)$ by the assumptions of the lemma for every $k = 1, \dots, K_j$. This implies that $\mathcal{I}(\mathcal{B}) \models_c \xi(E_1, \dots, E_m, \|t_1\|, \dots, \|t_n\|)$. □

Corollary 5.4. $\mathcal{I}(\mathcal{B}) \models_c S^b_L$.

Proof. From the definition of $\mathcal{I}(\mathcal{B})$ and the closure rules we get that $P(t_1, \dots, t_n) \in \mathcal{B}$ implies $(\|t_1\|, \dots, \|t_n\|) \in P^{\mathcal{I}(\mathcal{B})}$, $\neg P(t_1, \dots, t_n) \in \mathcal{B}$ implies $(\|t_1\|, \dots, \|t_n\|) \notin P^{\mathcal{I}(\mathcal{B})}$, $\nu_n(p, t_1, \dots, t_n) \in \mathcal{B}$ implies $(p, \|t_1\|, \dots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})}$, and $\neg\nu_n(p, t_1, \dots, t_n) \in \mathcal{B}$ implies $(p, \|t_1\|, \dots, \|t_n\|) \notin \nu_n^{\mathcal{I}(\mathcal{B})}$ for every constant predicate symbol P, $n = 0, \dots, N$, and primitive p of sort n. Thus, $\mathcal{I}(\mathcal{B}) \models_c S^b_L$ by Lemma 5.3. $\square$

Lemma 5.5. *$\mathcal{I}(\mathcal{B})$ reflects the branch $\mathcal{B}$.*

Proof. By simultaneous induction on the well-founded ordering $\prec$ induced by S_L we show that for all $n = 1, \dots, N$, for every E, and all domain ground terms $t_1, \dots, t_n$ (of sort $N+1$) in $\mathcal{B}$, we have that

- $(E, \|t_1\|, \dots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})}$ whenever $\nu_n(E, t_1, \dots, t_n) \in \mathcal{B}$, and
- $(E, \|t_1\|, \dots, \|t_n\|) \notin \nu_n^{\mathcal{I}(\mathcal{B})}$ whenever $\neg\nu_n(E, t_1, \dots, t_n) \in \mathcal{B}$.

We have the following two cases which correspond to the base case of the induction and to the induction step:

Case $E = p$. This case follows from the definition of $\mathcal{I}(\mathcal{B})$.

Case $E = \sigma(E_1, \dots, E_m)$. Suppose $\nu_n(E, t_1, \dots, t_n) \in \mathcal{B}$. Let ξ^F_+ be such that $E = F(F_1, \dots, F_m)$ for some $F_1, \dots, F_m$ and the Skolemised form of the corresponding ϕ^F_+ is as follows

$$\phi^F_+(p_1, \dots, p_m, x_1, \dots, x_n) \equiv \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk}(p_1, \dots, p_m, x_1, \dots, x_n).$$

Then by rule $\rho_+(\xi^F_+)$ there is a $j = 1, \dots, J$ such that, for all $k = 1, \dots, K_j$, the literals $\psi_{jk}(F_1, \dots, F_m, t_1, \dots, t_n)$ are in $\mathcal{B}$. Further, for every expression $E'(F_1, \dots, F_m)$ which occurs in $\psi_{jk}(F_1, \dots, F_m, t_1, \dots, t_n)$, where $k = 1, \dots, K_j$, we have $E'(F_1, \dots, F_m) \prec F(F_1, \dots, F_m) = E$. Thus, by the induction hypothesis, for every $k = 1, \dots, K_j$, $\mathcal{I}(\mathcal{B}) \models_c \psi_{jk}(F_1, \dots, F_m, \|t_1\|, \dots, \|t_n\|)$. Consequently, we have

$$\mathcal{I}(\mathcal{B}) \models_c \phi^F_+(F_1, \dots, F_m, \|t_1\|, \dots, \|t_n\|)$$

and, hence, $\mathcal{I}(\mathcal{B}) \models_c \Phi^E_+(\|t_1\|, \dots, \|t_n\|)$. By Lemma 5.3, $\mathcal{I}(\mathcal{B}) \models_c S^b_L \restriction \mathsf{sub}_\prec(E)$. Since $\mathcal{I}(\mathcal{B}) \models \forall S^0_L$, we obtain $\mathcal{I}(\mathcal{B}) \models_c \phi^\sigma(E_1, \dots, E_m, \|t_1\|, \dots, \|t_n\|)$ and, therefore, by the definition of $\mathcal{I}(\mathcal{B})$, we have $(E, \|t_1\|, \dots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})}$.

The second implication for negative literals is proved similarly. $\square$

As a consequence we obtain the following theorem.

Theorem 5.6 (Constructive completeness)**.** *Let T_L be a tableau calculus generated from a well-defined semantic specification S_L of a logic L. Then T_L is constructively complete.*

Proof. We only need to prove that $\mathcal{I}(\mathcal{B}) \models \forall S_L$. However, this follows from $\forall S^0_L, \forall S^b_L \models \forall S_L$ since $\mathcal{I}(\mathcal{B}) \models \forall S^b_L$ by Lemma 5.5 and Lemma 5.3. $\square$

6. Refining Rules by Turning Conclusions into Premises

Generally the degree of branching of the generated rules is higher than is necessary. Furthermore, representation of the generated rules involves the additional symbols of the language $\mathsf{FO}(\mathcal{L})$ creating a syntactic overhead which may not always be justified. To address these problems in this section, and the next, we introduce two techniques for refining the generated rules.

The first technique reduces the number of branches of a rule by constraining the rule with additional premises and deriving fewer conclusions. Suppose r is a tableau rule in a sound and constructively complete tableau calculus T_L. Suppose r has this form.

$$r \overset{\mathsf{def}}{=} \frac{X_0}{X_1 \mid \cdots \mid X_m}.$$

Let $X_i = \{\psi_1, \ldots, \psi_k\}$ be one of the denominators of the rule r for some $i \in \{1, \ldots, m\}$. Without loss of generality we assume that $i = 1$.

Consider the rules r_j with $j = 1, \ldots, k$ defined by

$$r_j \overset{\mathsf{def}}{=} \frac{X_0 \cup \{\sim\psi_j\}}{X_2 \mid \cdots \mid X_m}.$$

Each r_j is obtained from the rule r by removing the first denominator X_1 and adding the negation of one of the formulae in X_1 as a premise. We can drop any domain predication equalities from the numerator when they are not necessary.

Let r denote a rule in T_L. We denote by $\mathsf{ref}(r, T_L)$ the *refined tableau calculus* obtained from T_L by replacing the rule r with rules $r_1, \ldots, r_k$. It is clear that the calculus $\mathsf{ref}(r, T_L)$ is sound. In general, $\mathsf{ref}(r, T_L)$ is however not constructively complete. Nevertheless, analysis of the proofs of Lemma 5.5 and Lemma 5.3 shows that the following theorem is true.

Theorem 6.1. *Let T_L be a tableau calculus generated from a well-defined specification S_L of the logic L. Let r be the rule $X_0/X_1 \mid \cdots \mid X_m$ in T_L and suppose $\mathsf{ref}(r, T_L)$ is a refined version of T_L. Further, suppose $\mathcal{B}$ is an open branch in a $\mathsf{ref}(r, T_L)$-tableau derivation and for every set Y of $\mathcal{L}$-expressions from $\mathcal{B}$ the following holds.*

If all expressions in Y are reflected in $\mathcal{I}(\mathcal{B})$ then for every $E_1, \ldots, E_l \in Y$,

$$\begin{aligned} &X_0(E_1, \ldots, E_l, t_1, \ldots, t_n) \subseteq \mathcal{B} \text{ implies} \\ &\mathcal{I}(\mathcal{B}) \models X_i(E_1, \ldots, E_l, \|t_1\|, \ldots, \|t_n\|), \text{ for some } i = 1, \ldots, m. \end{aligned} \tag{†}$$

Then, $\mathcal{B}$ is reflected in $\mathcal{I}(\mathcal{B})$.

Roughly, condition (†) says that the replaced rule r is admissible in the model $\mathcal{I}(\mathcal{B})$ associated with $\mathcal{B}$ constructed using the refined calculus $\mathsf{ref}(r, T_L)$. An immediate consequence is the following.

Corollary 6.2. *If the condition of Theorem 6.1 holds for every open branch $\mathcal{B}$ of any $\mathsf{ref}(r, T_L)$-tableau derivation then the refined calculus $\mathsf{ref}(r, T_L)$ is constructively complete.*

Generalising this refinement to turning more than one denominator into premises is not difficult. Theorem 6.1 can be reformulated to accommodate this generalisation and the formulation of Corollary 6.2 does not change then.

We observe that the condition (†) is implied by the following condition:

$$\begin{array}{l}\text{if } X_0(E_1,\ldots,E_l,t_1,\ldots,t_n)\subseteq\mathcal{B} \text{ and } \mathcal{I}(\mathcal{B})\not\models X_1(E_1,\ldots,E_l,\|t_1\|,\ldots,\|t_n\|)\\ \text{then } X_i(E_1,\ldots,E_l,t_1,\ldots,t_n)\subseteq\mathcal{B},\ \text{for some } i=2,\ldots,m.\end{array}\qquad (\ddagger)$$

This follows by an induction argument on the well-founded ordering $\prec$.

For example, consider the generated rule for negative occurrences of the existential restriction operator given in Section 4.

$$\frac{\neg\nu_1(\exists r.p,x),\quad y\approx y}{\neg\nu_2(r,x,y)\ \mid\ \neg\nu_1(p,y)}$$

In most description logics it can be replaced with the more often seen rule:

$$\frac{\neg\nu_1(\exists r.p,x),\quad \nu_2(r,x,y)}{\neg\nu_1(p,y)}.$$

In such cases, condition (‡) has the following form.

$$\text{If } \neg\nu_1(\exists E.F,t)\in\mathcal{B} \text{ and } \mathcal{I}(\mathcal{B})\models\nu_2(E,t,t') \text{ then } \neg\nu_1(F,t')\in\mathcal{B}.$$

For description and modal logics such as $\mathcal{SO}$ the proof of this condition is typically part of the proof of the completeness theorem for the calculus which is standardly proved by induction on the well-founded relation $\prec$ (or equivalently, by induction on the way formulae are derived on a branch). For $\mathcal{SO}$ condition (‡) can be proved separately and implies that condition (†) is true for every branch of the refined tableau. Thus, this rule refinement preserves constructive completeness.

The default equality rules (given in Figure 2) added to every generated calculus are already in refined form. The rules that would be produced from the semantic specification of equality in Figure 1 have a different form. For example, the congruence rule

$$\frac{\nu_n(p,\overline{x}),\quad x_i\approx y_i}{\nu_n(p,x_1,\ldots,x_{i-1},y_i,x_{i+1},\ldots,x_n)}$$

is a refined form (obtained in two steps) of the following rule:

$$\frac{p\approx p,\quad x_1\approx x_1,\quad \ldots,\quad x_n\approx x_n,\quad y_i\approx y_i}{\neg\nu_n(p,\overline{x})\ \mid\ x_i\not\approx y_i\ \mid\ \nu_n(p,x_1,\ldots,x_{i-1},y_i,x_{i+1},\ldots,x_n)}.$$

Transitivity of a role r provides another example where rule refinement converts the rule

$$\frac{r\approx r,\quad x\approx x,\quad y\approx y,\quad z\approx z}{\neg\nu_2(r,x,y)\ \mid\ \neg\nu_2(r,y,z)\ \mid\ \nu_2(r,x,z)}$$

into the more familiar rule

$$\frac{\nu_2(r,x,y),\quad \nu_2(r,y,z)}{\nu_2(r,x,z)}.$$

Condition (†) holds in this case since it follows from the definition of $\mathcal{I}(\mathcal{B})$ that $\mathcal{I}(\mathcal{B})$ reflects all atomic formulae of the form $\nu_2(r,x,y)$ for any role symbol r in the branch $\mathcal{B}$.

As a negative example let us consider the possibility of replacing the rule for disjunction

$$\frac{\nu_1(p\sqcup q,x)}{\nu_1(p,x)\ \mid\ \nu_1(q,x)}(\sqcup)$$

by this rule.

$$\frac{\nu_1(p\sqcup q,x),\quad \neg\nu_1(p,x)}{\nu_1(q,x)}(\sqcup')$$

In KE tableau calculi this rule is used together with an analytic cut rule [16]. This raises the question whether a cut rule is essential for completeness and whether the $(\sqcup')$-rule alone would suffice instead of $(\sqcup)$.

Consider a tableau calculus T without any other rules to decompose positive occurrences of disjunctions except the standard rule $(\sqcup)$. Suppose T' is the calculus where the $(\sqcup)$-rule has been replaced by the $(\sqcup')$-rule. That is, $T' \overset{\mathsf{def}}{=} \mathsf{ref}((\sqcup), T)$. Examination reveals that condition (†) in Theorem 6.1 does not hold for T'. Given a formula $\nu_1(p \sqcup q, a)$, the branch $\mathcal{B}_0$ containing only $\nu_1(p \sqcup q, a)$ is fully expanded. The interpretation $\mathcal{I}(\mathcal{B}_0)$ constructed from $\mathcal{B}_0$ as defined in the previous section reflects the expressions p and q. The instantiation of the premise of the $(\sqcup)$-rule with the expressions p and q belongs to the branch $\mathcal{B}_0$, that is, $\nu_1(p \sqcup q, a) \in \mathcal{B}_0$, but $\mathcal{I}(\mathcal{B}_0) \not\models \nu_1(p, a)$ and $\mathcal{I}(\mathcal{B}_0) \not\models \nu_1(q, a)$. This means condition (†) fails for $\mathcal{B}_0$ and $Y \overset{\mathsf{def}}{=} \{p, q\}$.

The following example shows that T' is in fact incomplete. Let $\mathcal{B}_1$ be the branch with formulae $\nu_1(\neg p \sqcup \neg q, a), \nu_1(p, a), \nu_1(q, a)$. The branch is fully expanded, because the $(\sqcup')$-rule is not applicable. However the formulae are unsatisfiable. This is why KE tableau calculi typically contain an analytic cut rule for completeness.

7. Refinement based on Exploiting the Expressivity of the Logic

In some cases, the object logic L is expressive enough to represent its own semantics. For example, in the case of standard modal logics, any Kripke frame condition can be encoded if a slightly more expressive hybrid modal language is used [8, 9]. This phenomenon leads us to consider a second kind of refinement, where all 'holds' predicates $\nu_1, \ldots, \nu_N$ and additional predicates of $\mathit{FO}(\mathcal{L})$ are expressible via validity of special expressions of the primary sort (concepts) of the object logic.

What does it mean for logic L to be expressive enough to represent its own semantics? Assume that for every $n = 0, \ldots, N$ and every n-ary predicate constant P occurring in the specification S_L, there are expressions

$$C_n^+(p, \ell_1, \ldots, \ell_n), \quad C_n^-(p, \ell_1, \ldots, \ell_n), \quad D_P^+(\ell_1, \ldots, \ell_n) \quad \text{and} \quad D_P^-(\ell_1, \ldots, \ell_n)$$

of the primary sort (concepts), depending on variable p of sort n and individual variables $\ell_1, \ldots, \ell_n$ of sort 0, such that the following all hold.

$$\forall S_L \models \forall x \left(\nu_1(C_n^+(p, \ell_1, \ldots, \ell_n), x) \rightarrow \nu_n(p, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right) \tag{7.1}$$

$$\forall S_L \models \forall x \left(\nu_1(C_n^-(p, \ell_1, \ldots, \ell_n), x) \rightarrow \neg\nu_n(p, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right) \tag{7.2}$$

$$\forall S_L \models \forall x \left(\nu_1(D_P^+(\ell_1, \ldots, \ell_n), x) \rightarrow P(\nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right) \tag{7.3}$$

$$\forall S_L \models \forall x \left(\nu_1(D_P^-(\ell_1, \ldots, \ell_n), x) \rightarrow \neg P(\nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right) \tag{7.4}$$

It is worth noting that because the equality theory is included in the specification S_L the following also hold:

$$\forall S_L \models \forall x \left(\nu_1(D_\approx^+(\ell_1, \ell_2), x) \rightarrow \nu_0(\ell_1) \approx \nu_0(\ell_2)\right),$$

$$\forall S_L \models \forall x \left(\nu_1(D_\approx^-(\ell_1, \ell_2), x) \rightarrow \nu_0(\ell_1) \not\approx \nu_0(\ell_2)\right).$$

If there are expressions such that (7.1)–(7.4) are true it is possible to express all tableau rules in T_L in the object language $\mathcal{L}$ itself as follows.

Let ε be a one-to-one mapping of domain variables to variables of sort 0. Now we only need to replace every positive occurrence of $\nu_n(E, x_1, \dots, x_n)$ in T_L with the concept $C_n^+(E, \varepsilon(x_1), \dots, \varepsilon(x_n))$, every (negative) occurrence of $\neg\nu_n(E, x_1, \dots, x_n)$ in T_L with the concept $C_n^-(E, \varepsilon(x_1), \dots, \varepsilon(x_n))$. Similarly, all predicate constants P need to be replaced with occurrences of D_P^+ or D_P^- depending on the polarity of P. Then the domain sort $N+1$ of the meta-language $\mathsf{FO}(\mathcal{L})$ is reflected by the sort 0.

A small technical complication is caused by functions in $\mathsf{FO}(\mathcal{L})$ (Skolem functions and Skolem constants, in particular) occurring in the generated tableau rules. For them there may not be corresponding function symbols in the object language $\mathcal{L}$. This can be addressed by introducing new connectives f_g into $\mathcal{L}$ for every function g (including constants) of $\mathsf{FO}(\mathcal{L})$ so that for any $p_1, \dots, p_m, \ell_1, \dots, \ell_n$, the term $f_g(p_1, \dots, p_m, \ell_1, \dots, \ell_n)$ is of sort 0 and its semantics is defined by

$$\nu_0(f_g(p_1, \dots, p_m, \ell_1, \dots, \ell_n)) \overset{\mathsf{def}}{=} g(p_1, \dots, p_m, \nu_0(\ell_1), \dots, \nu_0(\ell_n)).$$

An alternative is to introduce unique, new individual constants (for every $p_1, \dots, p_m$, $\ell_1, \dots, \ell_n$) instead of new connectives.

If T is a tableau calculus for the logic L we denote by $\mathsf{tr}(T)$ the *refined tableau calculus* obtained by replacing every positive occurrence of $\nu_n(E, x_1, \dots, x_n)$ in T_L by the concept $C_n^+(E, \varepsilon(x_1), \dots, \varepsilon(x_n))$, every occurrence of $\neg\nu_n(E, x_1, \dots, x_n)$ by $C_n^-(E, \varepsilon(x_1), \dots, \varepsilon(x_n))$, every positive occurrence of a predicate constants P by D_P^+, every negative occurrence of a predicate constants P by D_P^-, and every function g with the new connective f_g.

Theorem 7.1. *Let T be a sound and complete tableau calculus for a logic L. If there are expressions such that (7.1)–(7.4) then $\mathsf{tr}(T)$ is sound and complete. If, in addition, T is constructively complete then $\mathsf{tr}(T)$ is also constructively complete for L.*

To illustrate the refinement introduced in this section we enrich the object language of $\mathcal{SO}$ with an additional connective. In particular, we add the colon connective :, with sort $(0, 1, 1)$, defined by:

$$\forall x \left(\nu_1(\ell : p, x) \equiv \nu_1(p, \nu_0(\ell))\right).$$

We also introduce connectives which correspond to Skolem functions into the object language.

This allows us to find object expressions for defining the predicates $\approx$, ν_1 and ν_2 in the language of the logic:

$$\begin{aligned} C_1^+(p, \ell) &\overset{\mathsf{def}}{=} \ell : p, & C_1^-(p, \ell) &\overset{\mathsf{def}}{=} \ell : \neg p, \\ C_2^+(r, \ell_1, \ell_2) &\overset{\mathsf{def}}{=} \ell_1 : \exists r.\{\ell_2\}, & C_2^-(r, \ell_1, \ell_2) &\overset{\mathsf{def}}{=} \ell_1 : \neg\exists r.\{\ell_2\}, \\ D_\approx^+(\ell_1, \ell_2) &\overset{\mathsf{def}}{=} \ell_1 : \{\ell_2\}, & D_\approx^-(\ell_1, \ell_2) &\overset{\mathsf{def}}{=} \ell_1 : \neg\{\ell_2\}. \end{aligned}$$

This means the notation of the tableau calculus can be refined and simplified. The refined and simplified rules are given in Figure 4. Comparing Figure 3 and Figure 4 we can see that the refined formulations of the rules

$$\frac{\nu_1(\{\ell\}, x)}{\nu_0(\ell) \approx x} \qquad \frac{\neg\nu_1(\{\ell\}, x)}{\nu_0(\ell) \not\approx x} \qquad \frac{\nu_1(\neg p, x)}{\neg\nu_1(p, x)}$$

Decomposition rules:

$$\frac{\ell : \neg\neg p}{\ell : p} \qquad \frac{\ell : (p \sqcup q)}{\ell : p \mid \ell : q} \qquad \frac{\ell : \neg(p \sqcup q)}{\ell : \neg p, \quad \ell : \neg q}$$

$$\frac{\ell : \exists r.p}{\ell : \exists r.\{f(r,p,\ell)\}, \quad f(r,p,\ell) : p} \qquad \frac{\ell : \neg\exists r.p, \quad \ell : \exists r.\{\ell'\}}{\ell' : \neg p}$$

Transitivity rule:

$$\frac{\ell : \exists r.\{\ell'\}, \quad \ell' : \exists r.\{\ell''\}}{\ell : \exists r.\{\ell''\}}$$

Equality congruence rules:

$$\frac{\ell : \{\ell'\}}{\ell' : \{\ell\}} \qquad \frac{\ell : \neg\{\ell'\}}{\ell' : \{\ell'\}} \qquad \frac{\ell : p}{\ell : \{\ell\}} \qquad \frac{\ell : \neg\exists r.\{\ell'\}}{\ell' : \{\ell'\}} \qquad \frac{\ell : p, \quad \ell : \{\ell'\}}{\ell' : p}$$

$$\frac{\ell : \exists r.\{\ell'\}, \quad \ell' : \{\ell''\}}{\ell : \exists r.\{\ell''\}} \qquad \frac{f(r,p,\ell) : \{f(r,p,\ell)\}, \quad \ell : \{\ell'\}}{f(r,p,\ell) : \{f(r,p,\ell')\}}$$

Closure rule:

$$\frac{\ell : p, \quad \ell : \neg p}{\bot}$$

Figure 4: Refined tableau rules for $\mathcal{SO}$.

are all redundant and can be removed from the refined tableau calculus since their premises coincide with the conclusions. Furthermore, the refined equality congruence rules equivalently reduce to a smaller set of rules. For instance, the refined rule of transitivity of the equality

$$\frac{\ell : \{\ell'\}, \quad \ell' : \{\ell''\}}{\ell : \{\ell''\}}$$

can be derived from the following rules.

$$\frac{\ell : \{\ell'\}}{\ell' : \{\ell\}} \qquad \frac{\ell : p, \quad \ell : \{\ell'\}}{\ell' : p}$$

Finally, the closure rule for equality is subsumed by the usual closure rule.

By Theorems 6.1 and 7.1 the rules in Figure 4 provide a sound and (constructively) complete labelled tableau calculus for the logic $\mathcal{SO}$.

8. Termination through Unrestricted Blocking

We say a tableau calculus T is *terminating* (for satisfiability) iff for every *finite* set of concepts $\mathcal{S}$ every closed tableau $T(\mathcal{S})$ is finite and every open tableau $T(\mathcal{S})$ has a finite open branch.

For some logics, for example, modal logic K, the synthesised tableau calculi are terminating but in general they are not. In order to guarantee termination, various blocking mechanisms have been developed. Generally one can distinguish between at least three kinds of blocking techniques: those that reuse domain terms, those that are based on case analysis over conjectured equality constraints between domain terms and equality reasoning, and specialised loop checking mechanisms. Approaches based on reusing domain terms have been used for minimal model generation for classical logic [11, 12]. Approaches based on conjectured equality constraints include [7, 26, 32]. Loop checking mechanisms are based on comparing sets of concepts (expressions of sort 1) labelled by the same domain terms (or individuals) with minimal equality reasoning and without explicitly conjectured equality constraints and backtracking. Several such loop checking mechanisms have been developed for different modal and description logics, but also hybrid logics and other logics [25, 5, 24, 10, 14].

In this section we adopt the unrestricted blocking mechanism of [32] to obtain terminating tableau calculi. An alternative that could also be used is blocking through reusing domain terms, but this would have required changing the rules of the calculus. Both unrestricted blocking and blocking through reuse of terms are less specialised and more generic than standard loop checking mechanisms.

Though introduced for deciding expressive description logics with role negation, the applicability of the unrestricted blocking mechanism is not limited to description logics [33]. It provides a powerful method for obtaining tableau decision procedures for logics with the effective finite model property (with respect to their semantics).

A logic L has the *effective finite model property* iff there is a computable function μ, with the set of all finite sets of concept expressions as domain and a subset of the set of natural numbers as range, such that the following holds: For every finite set of concept expressions $\mathcal{S}$, if $\mathcal{S}$ is satisfiable in an L-model then there is a finite L-model for $\mathcal{S}$ with the number of elements in the domain not exceeding $\mu(\mathcal{S})$.

The unrestricted blocking mechanism is based on adding the following rule, called the *unrestricted blocking* rule, to a sound and complete tableau calculus.

$$\frac{x \approx x, \quad y \approx y}{x \approx y \mid x \not\approx y}(\mathsf{ub})$$

In our context the idea is that the rule conjectures whether pairs of domain terms (of sort $N+1$ in $\mathit{FO}(\mathcal{L})$) on the current branch are equal or not. In the left branch two such terms are conjectured to be equal. If this leads to a contradiction then they cannot be equal, which is the information carried by the right branch. The rule is generally sound, thus adding it to any sound and (constructive) complete tableau calculus preserves soundness and (constructive) completeness.

For termination it is crucial to impose additional restrictions on the application of the rules in the tableau calculus that introduce new domain terms to the derivation. This is achieved by defining an ordering $<$ on terms and imposing conditions (c1) and (c2) below on the calculus.

In particular, let $<$ be an ordering of terms of the domain sort $N+1$ in the branch which is a linear extension of the order of the introduction of the terms during the derivation. That is, $t < t'$, whenever the first appearance of term t' in the branch is strictly later than the first appearance of term t. The mentioned conditions are:

(c1) If $t \approx t'$ appears in a branch and $t < t'$, then possible applications of any rules to formulae with the term t' producing new terms of the domain sort are not performed.

(c2) In every open branch there is some node from which point onwards before any application of any rules which produce new terms of the domain sort all possible applications of the (ub) rule have been performed.

Condition (c1) specifies that term-producing rules may only be applied to formulae where the domain terms are the smallest representatives in their equivalence classes. The positive rule for $\exists \cdot .\cdot$ is the only term-producing rule in the calculus for $\mathcal{SO}$. Condition (c2) says that at some point in a branch the unrestricted blocking rule has been applied exhaustively before the application of term-producing rules.

For a tableau calculus T we denote by $T + (\text{ub})$ a tableau calculus obtained from T by adding the above blocking mechanism based on the unrestricted blocking rule.

According to [33], one of the prerequisites for termination of the calculus $T + (\text{ub})$ is the subexpression property of T. Let $\preceq$ be a reflexive and transitive ordering on $\mathcal{L}$-expressions. Following [33], we say that a tableau calculus T is *compatible with* $\mathsf{sub}_{\preceq}$, or has the *subexpression property* with respect to $\preceq$, iff for every set of concepts $\mathcal{S}$, all $\mathcal{L}$-expressions occurring in the tableau derivation $T(\mathcal{S})$ belong to $\mathsf{sub}_{\preceq}(\mathcal{S})$.

Given a well-defined semantic specification S the process of construction of T from S described in Section 4 ensures that every rule of T is monotone with respect to the ordering $\prec$ induced by S. That is, every $\mathcal{L}$-expression in each conclusion of a rule is not greater with respect to $\prec$ than $\mathcal{L}$-expressions in the premises of the rule. Therefore, we can conclude that T has the subexpression property with respect to the reflexive closure of the ordering $\prec$. Thus:

Lemma 8.1. *Let $\preceq$ be a reflexive closure of the ordering $\prec$ induced by a well-defined semantic specification S. Then the tableau calculus T generated from S has the subexpression property with respect to $\preceq$.*

This property is a necessary condition for termination of the calculus enhanced by the unrestricted blocking rule mechanism [32, 33]. Another necessary condition for termination is finiteness of $\mathsf{sub}_{\preceq}$. The operator sub mapping sets of concepts to sets of expressions is *finite* iff $\mathsf{sub}(\mathcal{S})$ is finite for every finite set of concepts $\mathcal{S}$. By König's Infinity Lemma, $\mathsf{sub}_{\preceq}$ is finite whenever $\prec$ is well-founded and finitely branching. Therefore:

Lemma 8.2. *Let $\preceq$ be a reflexive closure of the ordering $\prec$ induced by a well-defined semantic specification S. If $S^+ \cup S^-$ is finite then the operator $\mathsf{sub}_{\preceq}$ is finite.*

Reformulating the main result in [33] in terms of the notation of this paper gives us:

Theorem 8.3. *Let L be a logic and T be a sound and constructively complete tableau calculus for a semantic specification S_L of the logic L. Then $T + (\text{ub})$ is sound and constructively complete for S_L. Furthermore, $T + (\text{ub})$ is terminating for L, if the following conditions all hold:*

(1) *There is a finite closure operator sub (defined on sets of concepts of the language of L) such that T is compatible with sub.*
(2) *L has the effective finite model property with respect to S_L.*

From this theorem and Theorems 5.1, 5.6, 6.1, 7.1 and Lemmas 8.1 and 8.2 it follows that the extensions of the generated and refined tableau calculi with unrestricted blocking are sound and (constructively) complete. Moreover, if it is known that the given logic has the effective finite model property with respect to a finite semantic specification then both extensions are terminating as well.

It is well known that $\mathcal{SO}$ has the effective finite model property with respect to $S_{\mathcal{SO}}$, and clearly $S_{\mathcal{SO}}$ has a finite number of statements. As a consequence, a terminating tableau calculus for $\mathcal{SO}$ is obtained if the calculus in Figure 4 is enhanced with the unrestricted blocking mechanism as described above. Using the refinements in Section 7 the unrestricted blocking rule can be transformed as follows.

$$\frac{\ell : \{\ell\}, \quad \ell' : \{\ell'\}}{\ell : \{\ell'\} \mid \ell : \neg\{\ell'\}}(\mathsf{ub}')$$

Let $\mathcal{T}_{\mathcal{SO}}$ be a tableau calculus comprising of the rules listed in Figure 4 and the rule (ub′).

Theorem 8.4. *The calculus $\mathcal{T}_{\mathcal{SO}}$ is sound and constructively complete for $\mathcal{SO}$. Furthermore, $\mathcal{T}_{\mathcal{SO}}$ is terminating provided that conditions (c1) and (c2) are both true for $\mathcal{T}_{\mathcal{SO}}$-derivations.*

In the calculus $\mathcal{T}_{\mathcal{SO}}$, the imposed conditions (c1) and (c2) are restrictions on applications of the rule

$$\frac{\ell : \exists r.p}{\ell : \exists r.\{f(r,p,\ell)\}, \quad f(r,p,\ell) : p}.$$

Following [33], the calculus $\mathcal{T}_{\mathcal{SO}}$ can be turned into a deterministic decision procedure using breadth-first search or depth-first search.

The calculus $\mathcal{T}_{\mathcal{SO}}$ presents a new terminating tableau calculus for $\mathcal{SO}$ or equivalent hybrid logics. The main difference to existing tableau approaches (in a similar style) for $\mathcal{SO}$ or equivalent hybrid logics is that the individuals (or nominals) are handled differently. To force termination typically either equality or subset ancestor loop checking is used, and often transitivity is handled by a propagation rule.

9. Synthesising Tableau Calculi for Intuitionistic Logic

We consider another example to illustrate the method. Propositional intuitionistic logic IPC is a logic where the 'holds' predicates $\nu_1, \ldots, \nu_N$ cannot be expressed in the language of the logic. It is non-Boolean and provides an example of a logic where the background theory interacts with the definitions of the connectives.

The language of intuitionistic logic is a one-sorted language defined over a countable set of propositional symbols $p, q, p_1, p_2, \ldots$, and the standard connectives are $\rightarrow, \vee, \wedge, \bot$. The semantic specification S_{IPC} in $\mathit{FO}(\mathcal{L})$ is given by (confer [28]):

Connective definitions:

$$\begin{aligned}
\forall x\ \big(\nu_1(\bot, x) &\equiv \bot\big)\\
\forall x\ \big(\nu_1(p \wedge q, x) &\equiv \nu_1(p,x) \wedge \nu_1(q,x)\big)\\
\forall x\ \big(\nu_1(p \vee q, x) &\equiv \nu_1(p,x) \vee \nu_1(q,x)\big)\\
\forall x\ \big(\nu_1(p \rightarrow q, x) &\equiv \forall y\ \big(R(x,y) \rightarrow (\nu_1(p,y) \rightarrow \nu_1(q,y))\big)\big)
\end{aligned}$$

Background theory:

$$\begin{aligned}
&\forall x\, R(x,x)\\
&\forall x \forall y\, (R(x,y) \wedge R(y,x) \rightarrow x \approx y)\\
&\forall x \forall y \forall z\, (R(x,y) \wedge R(y,z) \rightarrow R(x,z))\\
&\forall x \forall y\, \big(\nu_1(p,x) \wedge R(x,y) \rightarrow \nu_1(p,y)\big)
\end{aligned}$$

The connective definitions impose the usual requirements for truth of a formula in a world of an intuitionistic Kripke model. For instance, the definition of implication expresses in $\mathit{FO}(\mathcal{L})$ the property that an implication of q from p is true in a world x if and only if q is true in every successor of x whenever p is true in that successor. R is the domain predicate symbol representing a partial order, which is specified by the first three sentences of the

Decomposition rules:

$$\frac{\nu_1(\bot, x)}{\bot} \qquad \frac{\neg\nu_1(\bot, x)}{\neg\bot}$$

$$\frac{\nu_1(p \wedge q, x)}{\nu_1(p, x), \quad \nu_1(q, x)} \qquad \frac{\neg\nu_1(p \wedge q, x)}{\neg\nu_1(p, x) \mid \neg\nu_1(q, x)}$$

$$\frac{\nu_1(p \vee q, x)}{\nu_1(p, x) \mid \nu_1(q, x)} \qquad \frac{\neg\nu_1(p \vee q, x)}{\neg\nu_1(p, x), \quad \neg\nu_1(q, x)}$$

$$\frac{\nu_1(p \rightarrow q, x), \quad y \approx y}{\neg R(x, y) \mid \neg\nu_1(p, y) \mid \nu_1(q, y)}$$

$$\frac{\neg\nu_1(p \rightarrow q, x)}{R(x, f(p, q, x)), \quad \nu_1(p, f(p, q, x)), \quad \neg\nu_1(q, f(p, q, x))}$$

Theory rules:

$$\frac{x \approx x}{R(x, x)} \qquad \frac{x \approx x, \quad y \approx y}{\neg R(x, y) \mid \neg R(y, x) \mid x \approx y} \qquad \frac{x \approx x, \quad y \approx y, \quad z \approx z}{\neg R(x, y) \mid \neg R(y, z) \mid R(x, z)}$$

$$\frac{p \approx p, \quad x \approx x, \quad y \approx y}{\neg\nu_1(p, x) \mid \neg R(x, y) \mid \nu_1(p, y)}$$

Closure rules:

$$\frac{\nu_1(p, x), \quad \neg\nu_1(p, x)}{\bot} \qquad \frac{R(x, y), \quad \neg R(x, y)}{\bot}$$

Figure 5: Generated tableau rules for intuitionistic logic.

background theory. The last sentence in the background theory specifies monotonicity of the truth of formulae (of sort 1).

For intuitionistic logic the orderings $\prec_0$ and $\prec$ coincide. The ordering $\prec$ on subexpressions induced by the semantic definition of the connectives is the smallest ordering satisfying $E_1 \prec E_1\sigma E_2$ and $E_2 \prec E_1\sigma E_2$, for each $\sigma \in \{\rightarrow, \vee, \wedge\}$ and any intuitionistic formulae E_1 and E_2. That is, $\prec$ is the direct subexpression ordering on intuitionistic formulae. Thus, the closure operator $\mathsf{sub}_{\preceq}$ induced by the reflexive closure $\preceq$ of the ordering $\prec$ is finite.

The tableau rules generated from the specification S_{IPC} are those listed in Figure 5. Together with the equality rules of Figure 2, they form a calculus, which is sound and constructively complete for intuitionistic logic. This is a consequence of Theorems 5.1 and 5.6.

Refining the generated rules yields the rules listed in Figure 6. Using Theorem 6.1 we conclude that together with the equality rules these rules provide a sound and constructively complete tableau calculus for intuitionistic logic. We denote this calculus by T_{IPC}.

Similarly to the case of $\mathcal{SO}$, because intuitionistic logic has the effective finite model property, by Theorem 8.3 together with Lemmas 8.1 and 8.2, a terminating tableau calculus for IPC is obtained if the calculus T_{IPC} is enhanced with the unrestricted blocking mechanism.

Theorem 9.1. *The tableau calculus* $T_{\mathsf{IPC}} +$ (ub) *is sound, constructively complete and terminating for* IPC.

Following [33], $T_{\mathsf{IPC}} +$ (ub) can be turned into deterministic decision procedures for IPC using breadth-first search or depth-first search.

Decomposition rules:

$$\frac{\nu_1(\bot, x)}{\bot} \qquad \frac{\nu_1(p \wedge q, x)}{\nu_1(p, x), \quad \nu_1(q, x)} \qquad \frac{\neg\nu_1(p \wedge q, x)}{\neg\nu_1(p, x) \mid \neg\nu_1(q, x)}$$

$$\frac{\nu_1(p \vee q, x)}{\nu_1(p, x) \mid \nu_1(q, x)} \qquad \frac{\neg\nu_1(p \vee q, x)}{\neg\nu_1(p, x), \quad \neg\nu_1(q, x)} \qquad \frac{\nu_1(p \to q, x), \quad R(x, y)}{\neg\nu_1(p, y) \mid \nu_1(q, y)}$$

$$\frac{\neg\nu_1(p \to q, x)}{R(x, f(p, q, x)), \quad \nu_1(p, f(p, q, x)), \quad \neg\nu_1(q, f(p, q, x))}$$

Theory rules:

$$\frac{x \approx x}{R(x, x)} \qquad \frac{R(x, y), \quad R(y, x)}{x \approx y} \qquad \frac{R(x, y), \quad R(y, z)}{R(x, z)} \qquad \frac{\nu_1(p, x), \quad R(x, y)}{\nu_1(p, y)}$$

Closure rules:

$$\frac{\nu_1(p, x), \quad \neg\nu_1(p, x)}{\bot}$$

Figure 6: Refined tableau rules for intuitionistic logic.

10. Discussion and Conclusions

The method introduced in this paper automatically produces a sound and constructively complete tableau calculus from the semantic first-order specification of a many-sorted logic. The method is directly applicable to many non-classical logics and covers many types of ground tableau calculi commonly found in the literature.

On one hand, the formalisation is based on ideas used in the implementation of tableau decision procedures for modal and description logics in the MetTeL system [35, 36]. The MetTeL system provides a core for tableau derivations, which does not depend on a logical language. Due to this language flexibility, without any modification of the core code, the prover constructs (sound, complete, and terminating) tableau derivations for standard modal logics, superintuitionistic logics (via the Gödel translation), many description logics, as well as for logics of metrics and topology for which it was originally written. Termination is achieved via an implementation of generalisations of standard blocking mechanisms as well as the unrestricted blocking mechanism. This means that MetTeL provides an implementation of a tableau decision procedure for description logics with full support of the role negation operator, which can not currently be handled by other tableau-based description and modal logic theorem provers. On the other hand, the results of this paper provide the theoretical foundation for the correct behaviour of tableau algorithms implemented in MetTeL.

More importantly, the results can be viewed as providing a mathematical formalisation and generalisation of tableau development methodologies. The formalisation separates the creative part of tableau calculus development, which needs to be done by a human developer, and the automatic part of the development process, which can be left to an automated (currently first-order) prover and an automated tableau synthesiser. In general, there is no algorithm for checking that an arbitrarily given binary relation forms a well-founded ordering. Therefore the creative part is writing down the semantic specification of the object logic so that the conditions of well-foundedness of the orderings $\prec_0$ and $\prec$ hold. The automatic part deals with verification of the first-order conditions (wd1) and (wd3′), and the generation of tableau rules from the (well-defined) semantics provided by the developer.

For common modal and description logics conditions (wd1) and (wd3) are simple to check, even trivial in many cases. In fact, a developer usually implicitly formalises the logic's semantics S in such a way that $S = S^0 \cup S^b$. This is the case for almost all known logics. If the specification of the semantics satisfies $S = S^0 \cup S^b$ then conditions (wd1) and (wd3) hold trivially and the orderings $\prec_0$ and $\prec$ coincide. This means the ordering used for the specification of the semantics of the logical connectives (which is usually well-founded) is enough for tableau synthesis.

The following are examples of first-order definable logics, which all have a normalised and well-defined semantic specification according to the definitions in Section 3:

- most description logics, including $\mathcal{ALCO}$, $\mathcal{SO}$, $\mathcal{ALBO}$ [32], $\mathcal{SHOIQ}$ [24];
- most propositional modal logics, including K, K4, S4, KD45, S5;
- propositional intuitionistic logic [28] and many Kripke-complete propositional superintuitionistic logics;
- the logic of metric and topology [27].

This paper also presents a general method for proving (constructive) completeness of tableau calculi. In addition, the generated rules can be transformed to the rules with lower branching factors provided that condition (†) has been proved by induction on the ordering $\prec$ for the refined calculus.

With enough expressivity for representing the basics of the semantics within the logic it is possible to simplify the language of the tableau calculus. In this case, the obtained calculus is similar to tableau calculi for description logics with singleton concepts, but also hybrid modal logic [10] and labelled tableau calculi [18, 21]. Otherwise, the calculus has the same flavour as standard tableau calculi for intuitionistic logic, where every node in a tableau derivation is characterised by two complementary sets of true and false formulae (concepts).

That the generated calculi are constructively complete has the added advantage that models can be effectively generated from open, finished branches in tableau derivations. This means that the synthesised tableau calculi can be used for finding models. If the calculus includes the unrestricted blocking mechanism various strategies on the application of the unrestricted blocking rule can be employed for obtaining models with minimal domain sizes.

As case studies we considered tableau synthesis for propositional intuitionistic logic and the description logic $\mathcal{SO}$ with singleton concepts and transitive roles. We believe the approach is also applicable to most known, first-order definable modal and description logics including the ones mentioned above. Non first-order translatable logics such as propositional dynamic logic are currently beyond the scope of the method.

The tableau calculi generated are Smullyan-type tableau calculi, that is, ground semantic tableau calculi. We believe that other types of tableau calculi can be generated using the same techniques. We expect that generating unlabelled tableau calculi without explicit background predicates or domain terms will be possible, at least to some extent, but this is not immediate. One possibility would be to investigate if these can be derived as further refinements of the labelled tableau calculi generated by method presented in this paper. Such a line of investigation would be interesting and shed more light on the relationship between different kinds of tableau calculi. Exploiting the known relationships to other deduction methods we expect synthesis of non-tableau approaches is possible as well, but all this is future work.

Further investigations are needed to explore the extension of the framework to generate calculi based on propagation rules which incorporate frame correspondence properties into the definition of connectives to replace the theory rules for modal and description logics (for example, transitivity for the logic $\mathcal{SO}$). It is clear though that this is a much harder problem because guaranteeing completeness becomes more difficult. It is also clear that no results at the same level of generality as for the use of theory rules in this paper can be expected.

A future goal is to further reduce human involvement in the development of calculi by finding appropriate automatically verifiable conditions for refined calculi to be generated.

We plan to implement the methodology as an automatic generator of tableau calculi. This will give users the ability to obtain tableau calculi very easily and without needing to have relevant knowledge of tableau-based reasoning or experience in developing tableau calculi. Combined with a prover engineering platform such as LoTREC [19] or the Tableau Workbench [1] there is even the potential to build systems that would allow users to get implemented provers from the specification of logics. LoTREC and the Tableau Workbench are generic systems for building tableau-based theorem provers for non-classical logics. Currently they allow users to define tableau procedures by flexibly specifying the set of tableau rules, the search strategies, the blocking technique and the optimisation techniques to be used. This is then compiled into a specialised prover for the specified procedure. Enhanced with the tableau synthesis methodology, such systems could allow the user to define just the logic and produce an implemented prover for this logic.

References

[1] P. Abate and R. Goré. The Tableaux Work Bench. In M. C. Mayer and F. Pirri, eds, *Proceedings of the 12th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'03)*, vol. 2796 of *Lecture Notes in Computer Science*, pp. 230–236. Springer, 2003.

[2] R. Alenda, N. Olivetti, C. Schwind, and D. Tishkovsky. Tableau calculi for CSL over minspaces. In A. Dawar and H. Veith, eds, *Proceedings of the 19th Annual Conference of the European Association for Computer Science Logic (CSL'10)*, vol. 6247 of *Lecture Notes in Computer Science*, pp. 52–66. Springer, 2010.

[3] A. Avellone, P. Miglioli, U. Moscato, and M. Ornaghi. Generalized tableau systems for intemediate propositional logics. In D. Galmiche, ed., *Proceedings of the 6th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'97)*, vol. 1227 of *Lecture Notes in Computer Science*, pp. 43–61. Springer, 1997.

[4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2003.

[5] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[6] P. Balbiani, H. P. van Ditmarsch, A. Herzig, and T. De Lima. Tableaux for public announcement logic. *Journal of Logic and Computation*, 20(1):55–76, 2010.

[7] P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In U. Furbach and N. Shankar, eds, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, vol. 4130 of *Lecture Notes in Artificial Intelligence*, pp. 125–139. Springer, 2006.

[8] P. Blackburn, M. de Rijke, and V. Venema. *Modal Logic*. Cambridge University Press, 2001.

[9] P. Blackburn and J. Seligman. What are hybrid languages? In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, eds, *Advances in Modal Logic, Volume 1*, pp. 41–62. CSLI Publications, 1998.

[10] T. Bolander and P. Blackburn. Termination for hybrid tableaus. *Journal of Logic and Computation*, 17(3):517–554, 2007.

[11] F. Bry and R. Manthey. Proving finite satisfiability of deductive databases. In E. Börger, H. Kleine Büning, and M. M. Richter, eds, *Proceedings of the 1st Workshop on Computer Science Logic (CSL'87)*, vol. 329 of *Lecture Notes in Computer Science*, pp. 44–55. Springer, 1988.

[12] F. Bry and S. Torge. A deduction method complete for refutation and finite satisfiability. In J. Dix, L. Fariñas del Cerro, and U. Furbach, eds, *Proceedings of the 6th European Conference on Logics in Artificial Intelligence (JELIA'98)*, vol. 1489 of *Lecture Notes in Computer Science*, pp. 1–17. Springer, 1998.

[13] M. A. Castilho, L. Fariñas del Cerro, O. Gasquet, and A. Herzig. Modal tableaux with propagation rules and structural rules. *Fundamenta Informaticae*, 3–4(32):281–297, 1997.

[14] M. Cialdea Mayer and S. Cerrito. Ground and free-variable tableaux for variants of quantified modal logics. *Studia Logica*, 69:97–131, 2001.

[15] M. Cialdea Mayer and S. Cerrito. Nominal substitution at work with the global and converse modalities. In L. Beklemishev, V. Goranko, and V. Shehtman, eds, *Advances in Modal Logic*, vol. 8, pp. 57–74. College Publications, 2010.

[16] M. D'Agostino and M. Mondadori. The taming of the cut. Classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[17] L. Fariñas del Cerro and O. Gasquet. A general framework for pattern-driven modal tableaux. *Logic Journal of the IGPL*, 10(1):51–83, 2002.

[18] M. Fitting. *Proof methods for modal and intuitionistic logics*. Kluwer, 1983.

[19] O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical tableaux research engineering companion. In B. Beckert, ed., *Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'05)*, vol. 3702 of *Lecture Notes in Computer Science*, pp. 318–322. Springer, 2005.

[20] V. Goranko and D. Shkatov. Tableau-based decision procedure for full coalitional multiagent temporal-epistemic logic of linear time. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, eds, *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'09)*, pp. 969–976. IFAAMAS, 2009.

[21] R. Goré. Tableau methods for modal and temporal logics. In M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, eds, *Handbook of Tableau Methods*. Springer, 1999.

[22] A. Heuerding. *Sequent calculi for proof search in some modal logics*. PhD thesis, Universität Bern, 1998.

[23] I. Horrocks, U. Hustadt, U. Sattler, and R. A. Schmidt. Computational modal logic. In P. Blackburn, J. van Benthem, and F. Wolter, eds, *Handbook of Modal Logic*, pp. 181–245. Elsevier, 2007.

[24] I. Horrocks and U. Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, 2007.

[25] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Routledge, London, 1968.

[26] U. Hustadt and R. A. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In T. Dean, ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pp. 110–115. Morgan Kaufmann, 1999.

[27] U. Hustadt, D. Tishkovsky, F. Wolter, and M. Zakharyaschev. Automated reasoning about metric and topology (System description). In M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, eds, *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA'06)*, vol. 4160 of *Lecture Notes in Artificial Intelligence*, pp. 490–493. Springer, 2006.

[28] S. A. Kripke. Semantical analysis of intuitionistic logic I. In J. N. Crossley and M. A. E. Dummett, eds, *Formal Systems and Recursive Functions*, pp. 92–130. North-Holland, 1965.

[29] F. Massacci. Single step tableaux for modal logics: Computational properties, complexity and methodology. *Journal of Automated Reasoning*, 24(3):319–364, 2000.

[30] B. Motik, R. Shearer, and I. Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[31] R. A. Schmidt. Developing modal tableaux and resolution methods via first-order resolution. In G. Governatori, I. M. Hodkinson, and Y. Venema, eds, *Advances in Modal Logic, Volume 6*, pp. 1–26. College Publications, 2006.

[32] R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika,

D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, eds, *Proceedings of the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC'07)*, vol. 4825 of *Lecture Notes in Computer Science*, pp. 438–451. Springer, 2007.

[33] R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In A. Armando, P. Baumgartner, and G. Dowek, eds, *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, vol. 5195 of *Lecture Notes in Computer Science*, pp. 194–209. Springer, 2008.

[34] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. In M. Giese and A. Waaler, eds, *Proceedings of the 18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'09)*, vol. 5607 of *Lecture Notes in Artificial Intelligence*, pp. 310–324. Springer, 2009.

[35] D. Tishkovsky. MetTeL system. `http://www.mettel-prover.org`.

[36] D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. MetTeL: A tableau prover with logic-independent inference engine. In G. Metcalfe and K. Brünnler, eds, *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'11)*, Lecture Notes in Computer Science. Springer, 2011. To appear.

[37] M. Tzakova. Tableau calculi for hybrid logics. In N. V. Murray, ed., *Proceedings of the 8th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'99)*, vol. 1617 of *Lecture Notes in Computer Science*, pp. 278–292. Springer, 1999.